Introducción a
JavaScript

Javier Eguíluz Pérez

Introducción a
JavaScript

Javier Eguíluz Pérez

Sobre este libro...

-
- **Esta versión impresa se creó el 25 de marzo de 2009 y todavía está incompleta**. La versión más actualizada de los contenidos de este libro se puede encontrar en http://www.librosweb.es/javascript
- Si quieres aportar sugerencias, comentarios, críticas o informar sobre errores, puedes enviarnos un mensaje a **contacto@librosweb.es**

Capítulo 1. Introducción

1.1. ¿Qué es JavaScript?

JavaScript es un lenguaje de programación que se utiliza principalmente para crear páginas web dinámicas.

Una página web dinámica es aquella que incorpora efectos como texto que aparece y desaparece, animaciones, acciones que se activan al pulsar botones y ventanas con mensajes de aviso al usuario.

Técnicamente, JavaScript es un lenguaje de programación interpretado, por lo que no es necesario compilar los programas para ejecutarlos. En otras palabras, los programas escritos con JavaScript se pueden probar directamente en cualquier navegador sin necesidad de procesos intermedios.

A pesar de su nombre, JavaScript no guarda ninguna relación directa con el lenguaje de programación Java. Legalmente, JavaScript es una marca registrada de la empresa Sun Microsystems, como se puede ver en http://www.sun.com/suntrademarks/.

1.2. Breve historia

A principios de los años 90, la mayoría de usuarios que se conectaban a Internet lo hacían con módems a una velocidad máxima de 28.8 kbps. En esa época, empezaban a desarrollarse las primeras aplicaciones web y por tanto, las páginas web comenzaban a incluir formularios complejos.

Con unas aplicaciones web cada vez más complejas y una velocidad de navegación tan lenta, surgió la necesidad de un lenguaje de programación que se ejecutara en el navegador del usuario. De esta forma, si el usuario no rellenaba correctamente un formulario, no se le hacía esperar mucho tiempo hasta que el servidor volviera a mostrar el formulario indicando los errores existentes.

Brendan Eich, un programador que trabajaba en Netscape, pensó que podría solucionar este problema adaptando otras tecnologías existentes (como *ScriptEase*) al navegador Netscape Navigator 2.0, que iba a lanzarse en 1995. Inicialmente, Eich denominó a su lenguaje *LiveScript*.

Posteriormente, Netscape firmó una alianza con Sun Microsystems para el desarrollo del nuevo lenguaje de programación. Además, justo antes del lanzamiento Netscape decidió cambiar el nombre por el de JavaScript. La razón del cambio de nombre fue exclusivamente por marketing, ya que Java era la palabra de moda en el mundo informático y de Internet de la época.

La primera versión de JavaScript fue un completo éxito y Netscape Navigator 3.0 ya incorporaba la siguiente versión del lenguaje, la versión 1.1. Al mismo tiempo, Microsoft

lanzó JScript con su navegador Internet Explorer 3. JScript era una copia de JavaScript al que le cambiaron el nombre para evitar problemas legales.

Para evitar una guerra de tecnologías, Netscape decidió que lo mejor sería estandarizar el lenguaje JavaScript. De esta forma, en 1997 se envió la especificación JavaScript 1.1 al organismo ECMA (*European Computer Manufacturers Association*).

ECMA creó el comité TC39 con el objetivo de *"estandarizar de un lenguaje de script multiplataforma e independiente de cualquier empresa"*. El primer estándar que creó el comité TC39 se denominó **ECMA-262**, en el que se definió por primera vez el lenguaje ECMAScript.

Por este motivo, algunos programadores prefieren la denominación *ECMAScript* para referirse al lenguaje JavaScript. De hecho, JavaScript no es más que la implementación que realizó la empresa Netscape del estándar ECMAScript.

La organización internacional para la estandarización (ISO) adoptó el estándar ECMA-262 a través de su comisión IEC, dando lugar al estándar ISO/IEC-16262.

1.3. Especificaciones oficiales

ECMA ha publicado varios estándares relacionados con ECMAScript. En Junio de 1997 se publicó la primera edición del estándar ECMA-262. Un año después, en Junio de 1998 se realizaron pequeñas modificaciones para adaptarlo al estandar ISO/IEC-16262 y se creó la segunda edición.

La tercera edición del estándar ECMA-262 (publicada en Diciembre de 1999) es la versión que utilizan los navegadores actuales y se puede consultar gratuitamente en http://www.ecma-international.org/publications/standards/Ecma-262.htm

Actualmente se encuentra en desarrollo la cuarta versión de ECMA-262, que podría incluir novedades como paquetes, *namespaces*, definición explícita de clases, etc.

ECMA también ha definido varios estándares relacionados con ECMAScript, como el estándar ECMA-357, que define una extensión conocida como E4X y que permite la integración de JavaScript y XML.

1.4. Cómo incluir JavaScript en documentos XHTML

La integración de JavaScript y XHTML es muy flexible, ya que existen al menos tres formas para incluir código JavaScript en las páginas web.

1.4.1. Incluir JavaScript en el mismo documento XHTML

El código JavaScript se encierra entre etiquetas `<script>` y se incluye en cualquier parte del documento. Aunque es correcto incluir cualquier bloque de código en cualquier zona de la página, se recomienda definir el código JavaScript dentro de la cabecera del documento (dentro de la etiqueta `<head>`):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Ejemplo de código JavaScript en el propio documento</title>
<script type="text/javascript">
  alert("Un mensaje de prueba");
</script>
</head>

<body>
<p>Un párrafo de texto.</p>
</body>
</html>
```

Para que la página XHTML resultante sea válida, es necesario añadir el atributo `type` a la etiqueta `<script>`. Los valores que se incluyen en el atributo `type` están estandarizados y para el caso de JavaScript, el valor correcto es `text/javascript`.

Este método se emplea cuando se define un bloque pequeño de código o cuando se quieren incluir instrucciones específicas en un determinado documento HTML que completen las instrucciones y funciones que se incluyen por defecto en todos los documentos del sitio web.

El principal inconveniente es que si se quiere hacer una modificación en el bloque de código, es necesario modificar todas las páginas que incluyen ese mismo bloque de código JavaScript.

1.4.2. Definir JavaScript en un archivo externo

Las instrucciones JavaScript se pueden incluir en un archivo externo de tipo JavaScript que los documentos XHTML enlazan mediante la etiqueta `<script>`. Se pueden crear todos los archivos JavaScript que sean necesarios y cada documento XHTML puede enlazar tantos archivos JavaScript como necesite.

Ejemplo:

Archivo `codigo.js`

```
alert("Un mensaje de prueba");
```

Documento XHTML

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Ejemplo de código JavaScript en el propio documento</title>
<script type="text/javascript" src="/js/codigo.js"></script>
</head>

<body>
```

```
<p>Un párrafo de texto.</p>
</body>
</html>
```

Además del atributo `type`, este método requiere definir el atributo `src`, que es el que indica la URL correspondiente al archivo JavaScript que se quiere enlazar. Cada etiqueta `<script>` solamente puede enlazar un único archivo, pero en una misma página se pueden incluir tantas etiquetas `<script>` como sean necesarias.

Los archivos de tipo JavaScript son documentos normales de texto con la extensión `.js`, que se pueden crear con cualquier editor de texto como Notepad, Wordpad, EmEditor, UltraEdit, Vi, etc.

La principal ventaja de enlazar un archivo JavaScript externo es que se simplifica el código XHTML de la página, que se puede reutilizar el mismo código JavaScript en todas las páginas del sitio web y que cualquier modificación realizada en el archivo JavaScript se ve reflejada inmediatamente en todas las páginas XHTML que lo enlazan.

1.4.3. Incluir JavaScript en los elementos XHTML

Este último método es el menos utilizado, ya que consiste en incluir trozos de JavaScript dentro del código XHTML de la página:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Ejemplo de código JavaScript en el propio documento</title>
</head>

<body>
<p onclick="alert('Un mensaje de prueba')">Un párrafo de texto.</p>
</body>
</html>
```

El mayor inconveniente de este método es que *ensucia* innecesariamente el código XHTML de la página y complica el mantenimiento del código JavaScript. En general, este método sólo se utiliza para definir algunos eventos y en algunos otros casos especiales, como se verá más adelante.

1.5. Etiqueta noscript

Algunos navegadores no disponen de soporte completo de JavaScript, otros navegadores permiten bloquearlo parcialmente e incluso algunos usuarios bloquean completamente el uso de JavaScript porque creen que así navegan de forma más segura.

En estos casos, es habitual que si la página web requiere JavaScript para su correcto funcionamiento, se incluya un mensaje de aviso al usuario indicándole que debería activar JavaScript para disfrutar completamente de la página. El siguiente ejemplo

muestra una página web basada en JavaScript cuando se accede con JavaScript activado y cuando se accede con JavaScript completamente desactivado.

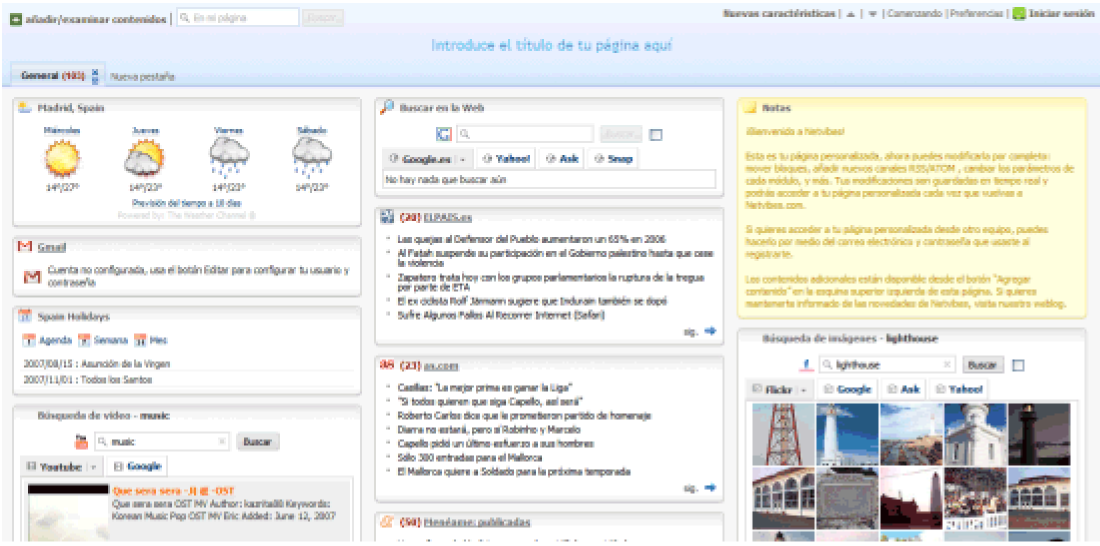


Figura 1.1. Imagen de www.Netvibes.com con JavaScript activado

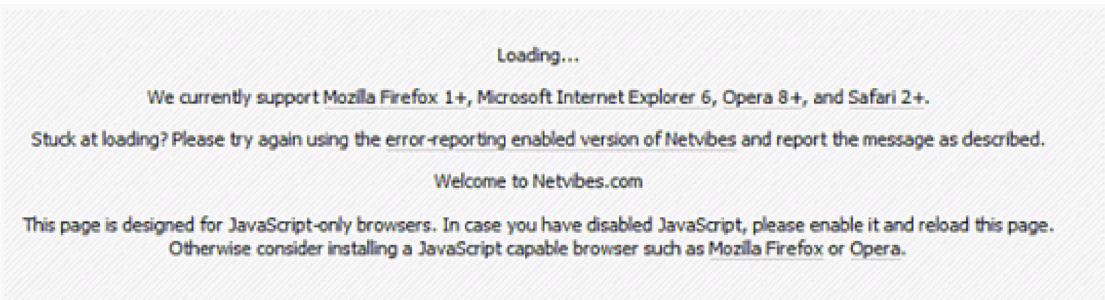


Figura 1.2. Imagen de www.Netvibes.com con JavaScript desactivado

El lenguaje HTML define la etiqueta <noscript> para mostrar un mensaje al usuario cuando su navegador no puede ejecutar JavaScript. El siguiente código muestra un ejemplo del uso de la etiqueta <noscript>:

```
<head> ... </head>
<body>
<noscript>
  <p>Bienvenido a Mi Sitio</p>
  <p>La página que estás viendo requiere para su funcionamiento el uso de
JavaScript.
Si lo has deshabilitado intencionadamente, por favor vuelve a activarlo.</p>
</noscript>
</body>
```

La etiqueta <noscript> se debe incluir en el interior de la etiqueta <body> (normalmente se incluye al principio de <body>). El mensaje que muestra <noscript> puede incluir cualquier elemento o etiqueta XHTML.

1.6. Glosario básico

Script: cada uno de los programas, aplicaciones o trozos de código creados con el lenguaje de programación JavaScript. Unas pocas líneas de código forman un script y un archivo de miles de líneas de JavaScript también se considera un script. A veces se traduce al español directamente como *"guión"*, aunque script es una palabra más adecuada y comúnmente aceptada.

Sentencia: cada una de las instrucciones que forman un script.

Palabras reservadas: son las palabras (en inglés) que se utilizan para construir las sentencias de JavaScript y que por tanto no pueden ser utilizadas libremente. Las palabras actualmente reservadas por JavaScript son: `break`, `case`, `catch`, `continue`, `default`, `delete`, `do`, `else`, `finally`, `for`, `function`, `if`, `in`, `instanceof`, `new`, `return`, `switch`, `this`, `throw`, `try`, `typeof`, `var`, `void`, `while`, `with`.

1.7. Sintaxis

La sintaxis de un lenguaje de programación se define como el conjunto de reglas que deben seguirse al escribir el código fuente de los programas para considerarse como correctos para ese lenguaje de programación.

La sintaxis de JavaScript es muy similar a la de otros lenguajes de programación como Java y C. Las normas básicas que definen la sintaxis de JavaScript son las siguientes:

- **No se tienen en cuenta los espacios en blanco y las nuevas líneas**: como sucede con XHTML, el intérprete de JavaScript ignora cualquier espacio en blanco sobrante, por lo que el código se puede ordenar de forma adecuada para entenderlo mejor (tabulando las líneas, añadiendo espacios, creando nuevas líneas, etc.)
- **Se distinguen las mayúsculas y minúsculas**: al igual que sucede con la sintaxis de las etiquetas y elementos XHTML. Sin embargo, si en una página XHTML se utilizan indistintamente mayúsculas y minúsculas, la página se visualiza correctamente, siendo el único problema la no validación de la página. En cambio, si en JavaScript se intercambian mayúsculas y minúsculas el script no funciona.
- **No se define el tipo de las variables**: al crear una variable, no es necesario indicar el tipo de dato que almacenará. De esta forma, una misma variable puede almacenar diferentes tipos de datos durante la ejecución del script.
- **No es necesario terminar cada sentencia con el carácter de punto y coma (`;`)**: en la mayoría de lenguajes de programación, es obligatorio terminar cada sentencia con el carácter `;`. Aunque JavaScript no obliga a hacerlo, es conveniente seguir la tradición de terminar cada sentencia con el carácter del punto y coma (`;`).
- **Se pueden incluir comentarios**: los comentarios se utilizan para añadir información en el código fuente del programa. Aunque el contenido de los

comentarios no se visualiza por pantalla, si que se envía al navegador del usuario junto con el resto del script, por lo que es necesario extremar las precauciones sobre la información incluida en los comentarios.

JavaScript define dos tipos de comentarios: los de una sola línea y los que ocupan varias líneas.

Ejemplo de comentario de una sola línea:

```
// a continuación se muestra un mensaje
alert("mensaje de prueba");
```

Los comentarios de una sola línea se definen añadiendo dos barras oblicuas (//) al principio de la línea.

Ejemplo de comentario de varias líneas:

```
/* Los comentarios de varias líneas son muy útiles
cuando se necesita incluir bastante información
en los comentarios */
alert("mensaje de prueba");
```

Los comentarios multilínea se definen encerrando el texto del comentario entre los símbolos /* y */.

1.8. Posibilidades y limitaciones

Desde su aparición, JavaScript siempre fue utilizado de forma masiva por la mayoría de sitios de Internet. La aparición de Flash disminuyó su popularidad, ya que Flash permitía realizar algunas acciones imposibles de llevar a cabo mediante JavaScript.

Sin embargo, la aparición de las aplicaciones AJAX programadas con JavaScript le ha devuelto una popularidad sin igual dentro de los lenguajes de programación web.

En cuanto a las limitaciones, JavaScript fue diseñado de forma que se ejecutara en un entorno muy limitado que permitiera a los usuarios confiar en la ejecución de los scripts.

De esta forma, los scripts de JavaScript no pueden comunicarse con recursos que no pertenezcan al mismo dominio desde el que se descargó el script. Los scripts tampoco pueden cerrar ventanas que no hayan abierto esos mismos scripts. Las ventanas que se crean no pueden ser demasiado pequeñas ni demasiado grandes ni colocarse fuera de la vista del usuario (aunque los detalles concretos dependen de cada navegador).

Además, los scripts no pueden acceder a los archivos del ordenador del usuario (ni en modo lectura ni en modo escritura) y tampoco pueden leer o modificar las preferencias del navegador.

Por último, si la ejecución de un script dura demasiado tiempo (por ejemplo por un error de programación) el navegador informa al usuario de que un script está consumiendo demasiados recursos y le da la posibilidad de detener su ejecución.

A pesar de todo, existen alternativas para poder saltarse algunas de las limitaciones anteriores. La alternativa más utilizada y conocida consiste en firmar digitalmente el script y solicitar al usuario el permiso para realizar esas acciones.

1.9. JavaScript y navegadores

Los navegadores más modernos disponibles actualmente incluyen soporte de JavaScript hasta la versión correspondiente a la tercera edición del estándar ECMA-262.

La mayor diferencia reside en el *dialecto* utilizado, ya que mientras Internet Explorer utiliza JScript, el resto de navegadores (Firefox, Opera, Safari, Konqueror) utilizan JavaScript.

1.10. JavaScript en otros entornos

La inigualable popularidad de JavaScript como lenguaje de programación de aplicaciones web se ha extendido a otras aplicaciones y otros entornos no relacionados con la web.

Herramientas como Adobe Acrobat permiten incluir código JavaScript en archivos PDF. Otras herramientas de Adobe como Flash y Flex utilizan ActionScript, un dialecto del mismo estándar de JavaScript.

Photoshop permite realizar pequeños scripts mediante JavaScript y la versión 6 de Java incluye un nuevo paquete (denominado `javax.script`) que permite integrar ambos lenguajes.

Por último, aplicaciones como Yahoo Widgets (http://widgets.yahoo.com/) y el Dashboard de Apple (http://www.apple.com/downloads/dashboard/) utilizan JavaScript para programar sus widgets.

Capítulo 2. El primer script

A continuación, se muestra un primer script sencillo pero completo:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>El primer script</title>

<script type="text/javascript">
  alert("Hola Mundo!");
</script>
</head>

<body>
<p>Esta página contiene el primer script</p>
</body>
</html>
```

En este ejemplo, el script se incluye como un bloque de código dentro de una página XHTML. Por tanto, en primer lugar se debe crear una página XHTML correcta que incluya la declaración del `DOCTYPE`, el atributo `xmlns`, las secciones `<head>` y `<body>`, la etiqueta `<title>`, etc.

Aunque el código del script se puede incluir en cualquier parte de la página, se recomienda incluirlo en la cabecera del documento, es decir, dentro de la etiqueta `<head>`.

A continuación, el código JavaScript se debe incluir entre las etiquetas `<script>...</script>`. Además, para que la página sea válida, es necesario definir el atributo `type` de la etiqueta `<script>`. Técnicamente, el atributo `type` se corresponde con "el tipo MIME", que es un estándar para identificar los diferentes tipos de contenidos. El "tipo MIME" correcto para JavaScript es `text/javascript`.

Una vez definida la zona en la que se incluirá el script, se escriben todas las sentencias que forman la aplicación. Este primer ejemplo es tan sencillo que solamente incluye una sentencia: `alert("Hola Mundo!");`.

La instrucción `alert()` es una de las utilidades que incluye JavaScript y permite mostrar un mensaje en la pantalla del usuario. Si se visualiza la página web de este primer script en cualquier navegador, automáticamente se mostrará una ventana con el mensaje `"Hola Mundo!"`.

A continuación se muestra el resultado de ejecutar el script en diferentes navegadores:



Figura 2.1. Mensaje mostrado con "alert()" en Internet Explorer

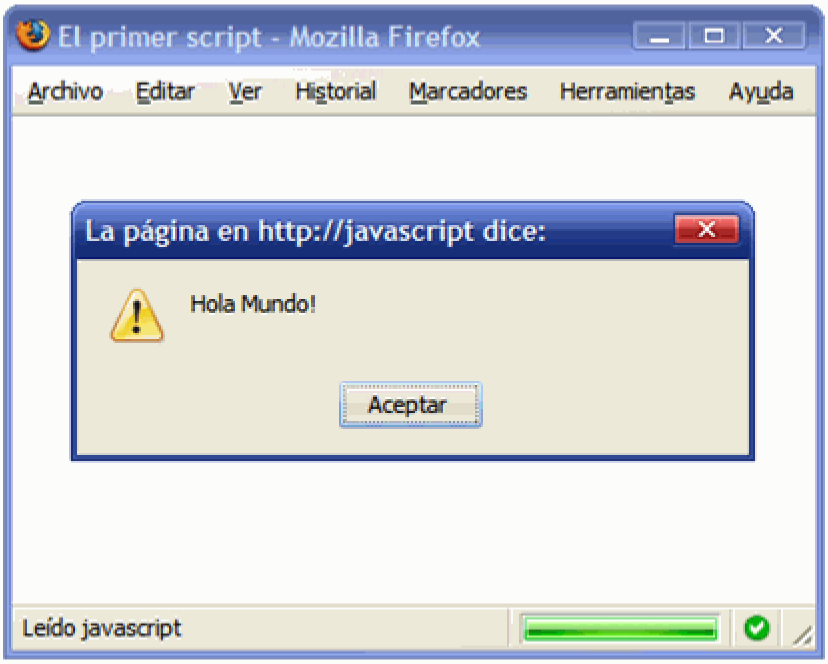


Figura 2.2. Mensaje mostrado con "alert()" en Firefox

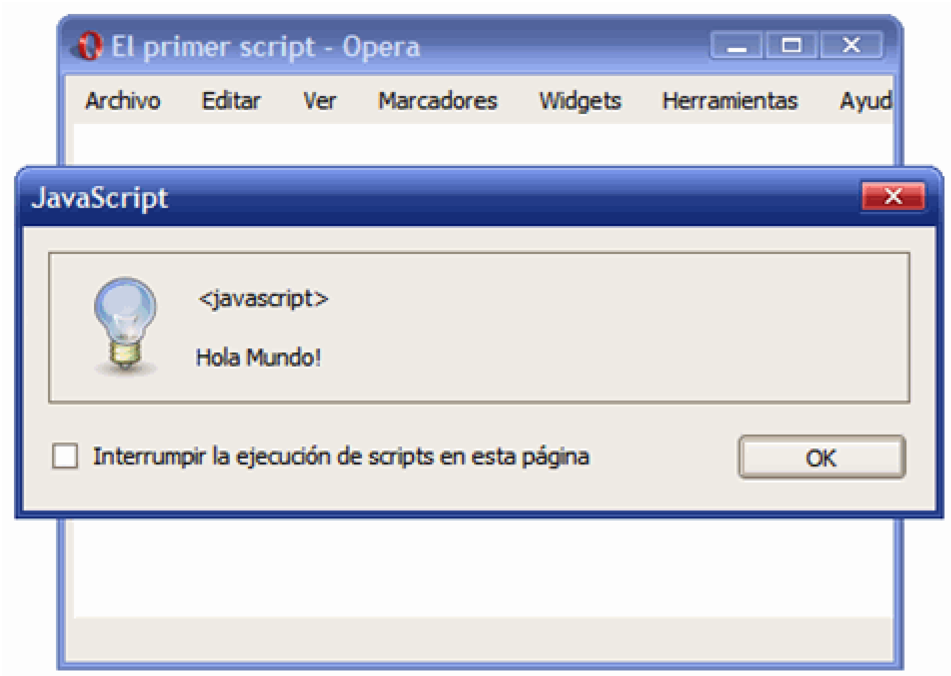


Figura 2.3. Mensaje mostrado con "alert()" en Opera

Como se puede observar en las imágenes anteriores, el funcionamiento de la utilidad `alert()` en los distintos navegadores es idéntico. Sin embargo, existen grandes diferencias visuales en la forma en la que se presentan los mensajes.

Ejercicio 1

Modificar el primer script para que:

1. Todo el código JavaScript se encuentre en un archivo externo llamado `codigo.js` y el script siga funcionando de la misma manera.
2. Después del primer mensaje, se debe mostrar otro mensaje que diga *"Soy el primer script"*
3. Añadir algunos comentarios que expliquen el funcionamiento del código
4. Añadir en la página XHTML un mensaje de aviso para los navegadores que no tengan activado el soporte de JavaScript

Capítulo 3. Programación básica

Antes de comenzar a desarrollar programas y utilidades con JavaScript, es necesario conocer los elementos básicos con los que se construyen las aplicaciones. Si ya sabes programar en algún lenguaje de programación, este capítulo te servirá para conocer la sintaxis específica de JavaScript.

Si nunca has programado, este capítulo explica en detalle y comenzando desde cero los conocimientos básicos necesarios para poder entender posteriormente la programación avanzada, que es la que se utiliza para crear las aplicaciones reales.

3.1. Variables

Las variables en los lenguajes de programación siguen una lógica similar a las variables utilizadas en otros ámbitos como las matemáticas. Una variable es un elemento que se emplea para almacenar y hacer referencia a otro valor. Gracias a las variables es posible crear *"programas genéricos"*, es decir, programas que funcionan siempre igual independientemente de los valores concretos utilizados.

De la misma forma que si en Matemáticas no existieran las variables no se podrían definir las ecuaciones y fórmulas, en programación no se podrían hacer programas realmente útiles sin las variables.

Si no existieran variables, un programa que suma dos números podría escribirse como:

```
resultado = 3 + 1
```

El *programa* anterior es tan poco útil que sólo sirve para el caso en el que el primer número de la suma sea el 3 y el segundo número sea el 1. En cualquier otro caso, el *programa* obtiene un resultado incorrecto.

Sin embargo, el programa se puede rehacer de la siguiente manera utilizando variables para almacenar y referirse a cada número:

```
numero_1 = 3
numero_2 = 1
resultado = numero_1 + numero_2
```

Los elementos `numero_1` y `numero_2` son **variables** que almacenan los valores que utiliza el programa. El resultado se calcula siempre en función del valor almacenado por las variables, por lo que este programa funciona correctamente para cualquier par de números indicado. Si se modifica el valor de las variables `numero_1` y `numero_2`, el programa sigue funcionando correctamente.

Las variables en JavaScript se crean mediante la palabra reservada `var`. De esta forma, el ejemplo anterior se puede realizar en JavaScript de la siguiente manera:

```
var numero_1 = 3;
var numero_2 = 1;
var resultado = numero_1 + numero_2;
```

La palabra reservada `var` solamente se debe indicar al definir por primera vez la variable, lo que se denomina **declarar** una variable. Cuando se utilizan las variables en el resto de instrucciones del script, solamente es necesario indicar su nombre. En otras palabras, en el ejemplo anterior sería un error indicar lo siguiente:

```
var numero_1 = 3;
var numero_2 = 1;
var resultado = var numero_1 + var numero_2;
```

Si cuando se declara una variable se le asigna también un valor, se dice que la variable ha sido **inicializada**. En JavaScript no es obligatorio inicializar las variables, ya que se pueden declarar por una parte y asignarles un valor posteriormente. Por tanto, el ejemplo anterior se puede rehacer de la siguiente manera:

```
var numero_1;
var numero_2;

numero_1 = 3;
numero_2 = 1;

var resultado = numero_1 + numero_2;
```

Una de las características más sorprendentes de JavaSript para los programadores habituados a otros lenguajes de programación es que tampoco es necesario declarar las variables. En otras palabras, se pueden utilizar variables que no se han definido anteriormente mediante la palabra reservada `var`. El ejemplo anterior también es correcto en JavaScript de la siguiente forma:

```
var numero_1 = 3;
var numero_2 = 1;
resultado = numero_1 + numero_2;
```

La variable `resultado` no está declarada, por lo que JavaScript crea una variable global (más adelante se verán las diferencias entre variables locales y globales) y le asigna el valor correspondiente. De la misma forma, también sería correcto el siguiente código:

```
numero_1 = 3;
numero_2 = 1;
resultado = numero_1 + numero_2;
```

En cualquier caso, se recomienda declarar todas las variables que se vayan a utilizar.

El nombre de una variable también se conoce como **identificador** y debe cumplir las siguientes normas:

- Sólo puede estar formado por letras, números y los símbolos `$` (dólar) y `_` (guión bajo).
- El primer carácter no puede ser un número.

Por tanto, las siguientes variables tienen nombres correctos:

```
var $numero1;
var _$letra;
var $$$otroNumero;
var $_a__$4;
```

Sin embargo, las siguientes variables tienen identificadores incorrectos:

```
var 1numero;       // Empieza por un número
var numero;1_123;  // Contiene un carácter ";"
```

3.2. Tipos de variables

Aunque todas las variables de JavaScript se crean de la misma forma (mediante la palabra reservada `var`), la forma en la que se les asigna un valor depende del tipo de valor que se quiere almacenar (números, textos, etc.)

3.2.1. Numéricas

Se utilizan para almacenar valores numéricos enteros (llamados *integer* en inglés) o decimales (llamados *float* en inglés). En este caso, el valor se asigna indicando directamente el número entero o decimal. Los números decimales utilizan el carácter `.` (punto) en vez de `,` (coma) para separar la parte entera y la parte decimal:

```
var iva = 16;        // variable tipo entero
var total = 234.65;  // variable tipo decimal
```

3.2.2. Cadenas de texto

Se utilizan para almacenar caracteres, palabras y/o frases de texto. Para asignar el valor a la variable, se encierra el valor entre comillas dobles o simples, para delimitar su comienzo y su final:

```
var mensaje = "Bienvenido a nuestro sitio web";
var nombreProducto = 'Producto ABC';
var letraSeleccionada = 'c';
```

En ocasiones, el texto que se almacena en las variables no es tan sencillo. Si por ejemplo el propio texto contiene comillas simples o dobles, la estrategia que se sigue es la de encerrar el texto con las comillas (simples o dobles) que no utilice el texto:

```
/* El contenido de texto1 tiene comillas simples, por lo que
    se encierra con comillas dobles */
var texto1 = "Una frase con 'comillas simples' dentro";

/* El contenido de texto2 tiene comillas dobles, por lo que
    se encierra con comillas simples */
var texto2 = 'Una frase con "comillas dobles" dentro';
```

No obstante, a veces las cadenas de texto contienen tanto comillas simples como dobles. Además, existen otros caracteres que son difíciles de incluir en una variable de texto (tabulador, `ENTER`, etc.) Para resolver estos problemas, JavaScript define un mecanismo

para incluir de forma sencilla caracteres especiales y problemáticos dentro de una cadena de texto.

El mecanismo consiste en sustituir el carácter problemático por una combinación simple de caracteres. A continuación se muestra la tabla de conversión que se debe utilizar:

Si se quiere incluir...	**Se debe incluir...**
Una nueva línea	`\n`
Un tabulador	`\t`
Una comilla simple	`\'`
Una comilla doble	`\"`
Una barra inclinada	`\\`

De esta forma, el ejemplo anterior que contenía comillas simples y dobles dentro del texto se puede rehacer de la siguiente forma:

```
var texto1 = 'Una frase con \'comillas simples\' dentro';

var texto2 = "Una frase con \"comillas dobles\" dentro";
```

Este mecanismo de JavaScript se denomina *"mecanismo de escape"* de los caracteres problemáticos, y es habitual referirse a que los caracteres han sido *"escapados"*.

Ejercicio 2

Modificar el primer script del capítulo anterior para que:

1. El mensaje que se muestra al usuario se almacene en una variable llamada `mensaje` y el funcionamiento del script sea el mismo.
2. El mensaje mostrado sea el de la siguiente imagen:

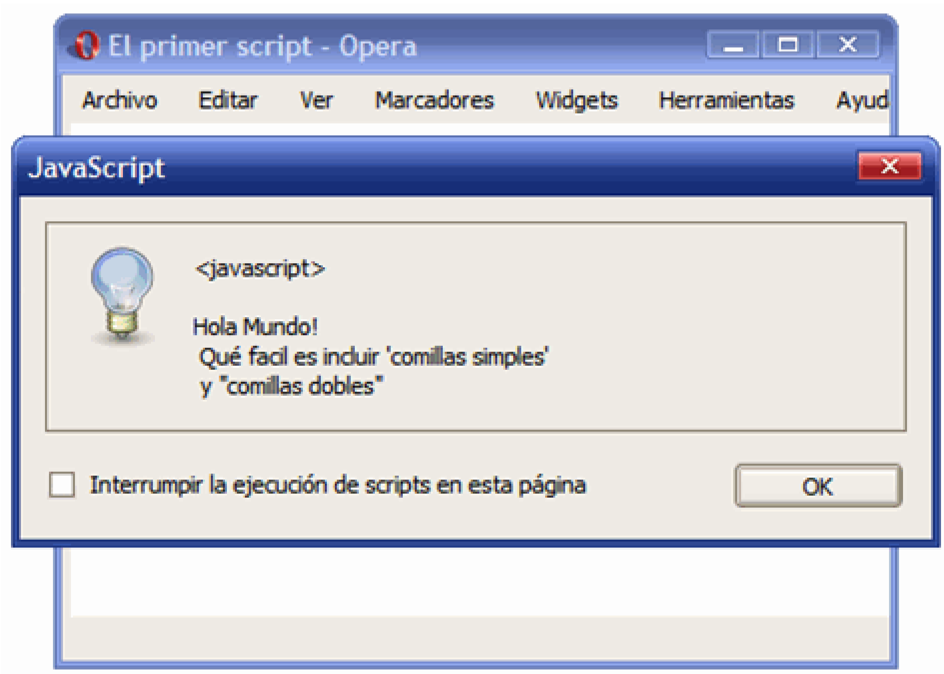


Figura 3.1. Nuevo mensaje que debe mostrar el script

3.2.3. Arrays

En ocasiones, a los arrays se les llama vectores, matrices e incluso *arreglos*. No obstante, el término array es el más utilizado y es una palabra comúnmente aceptada en el entorno de la programación.

Un array es una colección de variables, que pueden ser todas del mismo tipo o cada una de un tipo diferente. Su utilidad se comprende mejor con un ejemplo sencillo: si una aplicación necesita manejar los días de la semana, se podrían crear siete variables de tipo texto:

```
var dia1 = "Lunes";
var dia2 = "Martes";
...
var dia7 = "Domingo";
```

Aunque el código anterior no es incorrecto, sí que es poco eficiente y complica en exceso la programación. Si en vez de los días de la semana se tuviera que guardar el nombre de los meses del año, el nombre de todos los países del mundo o las mediciones diarias de temperatura de los últimos 100 años, se tendrían que crear decenas o cientos de variables.

En este tipo de casos, se pueden agrupar todas las variables relacionadas en una colección de variables o array. El ejemplo anterior se puede rehacer de la siguiente forma:

```
var dias = ["Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado",
"Domingo"];
```

Ahora, una única variable llamada `dias` almacena todos los valores relacionados entre sí, en este caso los días de la semana. Para definir un array, se utilizan los caracteres `[` y `]` para delimitar su comienzo y su final y se utiliza el carácter `,` (coma) para separar sus elementos:

```
var nombre_array = [valor1, valor2, ..., valorN];
```

Una vez definido un array, es muy sencillo acceder a cada uno de sus elementos. Cada elemento se accede indicando su posición dentro del array. La única complicación, que es responsable de muchos errores cuando se empieza a programar, es que las posiciones de los elementos empiezan a contarse en el `0` y no en el `1`:

```
var diaSeleccionado = dias[0];    // diaSeleccionado = "Lunes"
var otroDia = dias[5];            // otroDia = "Sábado"
```

En el ejemplo anterior, la primera instrucción quiere obtener el primer elemento del array. Para ello, se indica el nombre del array y entre corchetes la posición del elemento dentro del array. Como se ha comentado, las posiciones se empiezan a contar en el `0`, por lo que el primer elemento ocupa la posición `0` y se accede a el mediante `dias[0]`.

El valor `dias[5]` hace referencia al elemento que ocupa la sexta posición dentro del array `dias`. Como las posiciones empiezan a contarse en `0`, la posición `5` hace referencia al sexto elemento, en este caso, el valor `Sábado`.

Ejercicio 3

Crear un array llamado `meses` y que almacene el nombre de los doce meses del año. Mostrar por pantalla los doce nombres utilizando la función `alert()`.

3.2.4. Booleanos

Las variables de tipo *boolean* o *booleano* también se conocen con el nombre de variables de tipo lógico. Aunque para entender realmente su utilidad se debe estudiar la programación avanzada con JavaScript del siguiente capítulo, su funcionamiento básico es muy sencillo.

Una variable de tipo *boolean* almacena un tipo especial de valor que solamente puede tomar dos valores: `true` (verdadero) o `false` (falso). No se puede utilizar para almacenar números y tampoco permite guardar cadenas de texto.

Los únicos valores que pueden almacenar estas variables son `true` y `false`, por lo que no pueden utilizarse los valores `verdadero` y `falso`. A continuación se muestra un par de variables de tipo *booleano*:

```
var clienteRegistrado = false;
var ivaIncluido = true;
```

3.3. Operadores

Las variables por sí solas son de poca utilidad. Hasta ahora, sólo se ha visto cómo crear variables de diferentes tipos y cómo mostrar su valor mediante la función `alert()`. Para hacer programas realmente útiles, son necesarias otro tipo de herramientas.

Los operadores permiten manipular el valor de las variables, realizar operaciones matemáticas con sus valores y comparar diferentes variables. De esta forma, los operadores permiten a los programas realizar cálculos complejos y tomar decisiones lógicas en función de comparaciones y otros tipos de condiciones.

3.3.1. Asignación

El operador de asignación es el más utilizado y el más sencillo. Este operador se utiliza para guardar un valor específico en una variable. El símbolo utilizado es = (no confundir con el operador == que se verá más adelante):

```
var numero1 = 3;
```

A la izquierda del operador, siempre debe indicarse el nombre de una variable. A la derecha del operador, se pueden indicar variables, valores, condiciones lógicas, etc:

```
var numero1 = 3;
var numero2 = 4;

/* Error, la asignación siempre se realiza a una variable,
    por lo que en la izquierda no se puede indicar un número */
5 = numero1;

// Ahora, la variable numero1 vale 5
numero1 = 5;

// Ahora, la variable numero1 vale 4
numero1 = numero2;
```

3.3.2. Incremento y decremento

Estos dos operadores solamente son válidos para las variables numéricas y se utilizan para incrementar o decrementar en una unidad el valor de una variable.

Ejemplo:

```
var numero = 5;
++numero;
alert(numero);  // numero = 6
```

El operador de incremento se indica mediante el prefijo ++ en el nombre de la variable. El resultado es que el valor de esa variable se incrementa en una unidad. Por tanto, el anterior ejemplo es equivalente a:

```
var numero = 5;
numero = numero + 1;
alert(numero);  // numero = 6
```

De forma equivalente, el operador decremento (indicado como un prefijo `--` en el nombre de la variable) se utiliza para decrementar el valor de la variable:

```
var numero = 5;
--numero;
alert(numero);  // numero = 4
```

El anterior ejemplo es equivalente a:

```
var numero = 5;
numero = numero - 1;
alert(numero);  // numero = 4
```

Los operadores de incremento y decremento no solamente se pueden indicar como prefijo del nombre de la variable, sino que también es posible utilizarlos como sufijo. En este caso, su comportamiento es similar pero muy diferente. En el siguiente ejemplo:

```
var numero = 5;
numero++;
alert(numero);  // numero = 6
```

El resultado de ejecutar el script anterior es el mismo que cuando se utiliza el operador `++numero`, por lo que puede parecer que es equivalente indicar el operador ++ delante o

detrás del identificador de la variable. Sin embargo, el siguiente ejemplo muestra sus diferencias:

```
var numero1 = 5;
var numero2 = 2;
numero3 = numero1++ + numero2;
// numero3 = 7, numero1 = 6

var numero1 = 5;
var numero2 = 2;
numero3 = ++numero1 + numero2;
// numero3 = 8, numero1 = 6
```

Si el operador ++ se indica como prefijo del identificador de la variable, su valor se incrementa **antes** de realizar cualquier otra operación. Si el operador ++ se indica como sufijo del identificador de la variable, su valor se incrementa **después** de ejecutar la sentencia en la que aparece.

Por tanto, en la instrucción `numero3 = numero1++ + numero2;`, el valor de `numero1` se incrementa después de realizar la operación (primero se suma y `numero3` vale `7`, después se incrementa el valor de `numero1` y vale `6`). Sin embargo, en la instrucción `numero3 = ++numero1 + numero2;`, en primer lugar se incrementa el valor de `numero1` y después se realiza la suma (primero se incrementa `numero1` y vale `6`, después se realiza la suma y `numero3` vale `8`).

3.3.3. Lógicos

Los operadores lógicos son imprescindibles para realizar aplicaciones complejas, ya que se utilizan para tomar decisiones sobre las instrucciones que debería ejecutar el programa en función de ciertas condiciones.

El resultado de cualquier operación que utilice operadores lógicos siempre es un valor lógico o *booleano*.

3.3.3.1. Negación

Uno de los operadores lógicos más utilizados es el de la negación. Se utiliza para obtener el valor contrario al valor de la variable:

```
var visible = true;
alert(!visible);  // Muestra "false" y no "true"
```

La negación lógica se obtiene prefijando el símbolo `!` al identificador de la variable. El funcionamiento de este operador se resume en la siguiente tabla:

variable	**!variable**
`true`	`false`
`false`	`true`

Si la variable original es de tipo *booleano*, es muy sencillo obtener su negación. Sin embargo, ¿qué sucede cuando la variable es un número o una cadena de texto? Para obtener la negación en este tipo de variables, se realiza en primer lugar su conversión a un valor *booleano*:

- Si la variable contiene un número, se transforma en `false` si vale `0` y en `true` para cualquier otro número (positivo o negativo, decimal o entero).
- Si la variable contiene una cadena de texto, se transforma en `false` si la cadena es vacía (`""`) y en `true` en cualquier otro caso.

```
var cantidad = 0;
vacio = !cantidad;  // vacio = true

cantidad = 2;
vacio = !cantidad;  // vacio = false

var mensaje = "";
mensajeVacio = !mensaje;  // mensajeVacio = true

mensaje = "Bienvenido";
mensajeVacio = !mensaje;  // mensajeVacio = false
```

3.3.3.2. AND

La operación lógica `AND` obtiene su resultado combinando dos valores booleanos. El operador se indica mediante el símbolo `&&` y su resultado solamente es `true` si los dos operandos son `true`:

variable1	**variable2**	**variable1 && variable2**
`true`	`true`	`true`
`true`	`false`	`false`
`false`	`true`	`false`
`false`	`false`	`false`

```
var valor1 = true;
var valor2 = false;
resultado = valor1 && valor2; // resultado = false

valor1 = true;
valor2 = true;
resultado = valor1 && valor2; // resultado = true
```

3.3.3.3. OR

La operación lógica `OR` también combina dos valores booleanos. El operador se indica mediante el símbolo `||` y su resultado es `true` si alguno de los dos operandos es `true`:

variable1	**variable2**	**variable1 || variable2**
`true`	`true`	`true`

true	false	true
false	true	true
false	false	false

```
var valor1 = true;
var valor2 = false;
resultado = valor1 || valor2; // resultado = true

valor1 = false;
valor2 = false;
resultado = valor1 || valor2; // resultado = false
```

3.3.4. Matemáticos

JavaScript permite realizar manipulaciones matemáticas sobre el valor de las variables numéricas. Los operadores definidos son: suma (+), resta (-), multiplicación (*) y división (/). Ejemplo:

```
var numero1 = 10;
var numero2 = 5;

resultado = numero1 / numero2;  // resultado = 2
resultado = 3 + numero1;        // resultado = 13
resultado = numero2 – 4;        // resultado = 1
resultado = numero1 * numero 2; // resultado = 50
```

Además de los cuatro operadores básicos, JavaScript define otro operador matemático que no es sencillo de entender cuando se estudia por primera vez, pero que es muy útil en algunas ocasiones.

Se trata del operador *"módulo"*, que calcula el resto de la división entera de dos números. Si se divide por ejemplo `10` y `5`, la división es exacta y da un resultado de `2`. El resto de esa división es `0`, por lo que `módulo de 10 y 5` es igual a `0`.

Sin embargo, si se divide `9` y `5`, la división no es exacta, el resultado es `1` y el resto `4`, por lo que `módulo de 9 y 5` es igual a `4`.

El operador módulo en JavaScript se indica mediante el símbolo `%`, que no debe confundirse con el cálculo del porcentaje:

```
var numero1 = 10;
var numero2 = 5;
resultado = numero1 % numero2; // resultado = 0

numero1 = 9;
numero2 = 5;
resultado = numero1 % numero2; // resultado = 4
```

Los operadores matemáticos también se pueden combinar con el operador de asignación para abreviar su notación:

```
var numero1 = 5;
numero1 += 3;  // numero1 = numero1 + 3 = 8
numero1 -= 1;  // numero1 = numero1 - 1 = 4
numero1 *= 2;   // numero1 = numero1 * 2 = 10
numero1 /= 5;   // numero1 = numero1 / 5 = 1
numero1 %= 4;   // numero1 = numero1 % 4 = 1
```

3.3.5. Relacionales

Los operadores relacionales definidos por JavaScript son idénticos a los que definen las matemáticas: mayor que (>), menor que (<), mayor o igual (>=), menor o igual (<=), igual que (==) y distinto de (!=).

Los operadores que relacionan variables son imprescindibles para realizar cualquier aplicación compleja, como se verá en el siguiente capítulo de programación avanzada. El resultado de todos estos operadores siempre es un valor booleano:

```
var numero1 = 3;
var numero2 = 5;
resultado = numero1 > numero2; // resultado = false
resultado = numero1 < numero2; // resultado = true

numero1 = 5;
numero2 = 5;
resultado = numero1 >= numero2; // resultado = true
resultado = numero1 <= numero2; // resultado = true
resultado = numero1 == numero2; // resultado = true
resultado = numero1 != numero2; // resultado = false
```

Se debe tener especial cuidado con el operador de igualdad (==), ya que es el origen de la mayoría de errores de programación, incluso para los usuarios que ya tienen cierta experiencia desarrollando scripts. El operador == se utiliza para comparar el valor de dos variables, por lo que es muy diferente del operador =, que se utiliza para asignar un valor a una variable:

```
// El operador "=" asigna valores
var numero1 = 5;
resultado = numero1 = 3;  // numero1 = 3 y resultado = 3

// El operador "==" compara variables
var numero1 = 5;
resultado = numero1 == 3; // numero1 = 5 y resultado = false
```

Los operadores relacionales también se pueden utilizar con variables de tipo cadena de texto:

```
var texto1 = "hola";
var texto2 = "hola";
var texto3 = "adios";

resultado = texto1 == texto3; // resultado = false
resultado = texto1 != texto2; // resultado = false
resultado = texto3 >= texto2; // resultado = false
```

Cuando se utilizan cadenas de texto, los operadores "mayor que" (`>`) y "menor que" (`<`) siguen un razonamiento no intuitivo: se compara letra a letra comenzando desde la izquierda hasta que se encuentre una diferencia entre las dos cadenas de texto. Para determinar si una letra es mayor o menor que otra, las mayúsculas se consideran menores que las minúsculas y las primeras letras del alfabeto son menores que las últimas (`a` es menor que `b`, `b` es menor que `c`, `A` es menor que `a`, etc.)

Ejercicio 4

A partir del siguiente array que se proporciona: `var valores = [true, 5, false, "hola", "adios", 2];`

1. Determinar cual de los dos elementos de texto es mayor
2. Utilizando exclusivamente los dos valores booleanos del array, determinar los operadores necesarios para obtener un resultado `true` y otro resultado `false`
3. Determinar el resultado de las cinco operaciones matemáticas realizadas con los dos elementos numéricos

3.4. Estructuras de control de flujo

Los programas que se pueden realizar utilizando solamente variables y operadores son una simple sucesión lineal de instrucciones básicas.

Sin embargo, no se pueden realizar programas que muestren un mensaje si el valor de una variable es igual a un valor determinado y no muestren el mensaje en el resto de casos. Tampoco se puede repetir de forma eficiente una misma instrucción, como por ejemplo sumar un determinado valor a todos los elementos de un array.

Para realizar este tipo de programas son necesarias las **estructuras de control de flujo**, que son instrucciones del tipo *"si se cumple esta condición, hazlo; si no se cumple, haz esto otro"*. También existen instrucciones del tipo *"repite esto mientras se cumpla esta condición"*.

Si se utilizan estructuras de control de flujo, los programas dejan de ser una sucesión lineal de instrucciones para convertirse en programas *inteligentes* que pueden tomar decisiones en función del valor de las variables.

3.4.1. Estructura if

La estructura más utilizada en JavaScript y en la mayoría de lenguajes de programación es la estructura `if`. Se emplea para tomar decisiones en función de una condición. Su definición formal es:

```
if(condicion) {
  ...
}
```

Si la condición se cumple (es decir, si su valor es `true`) se ejecutan todas las instrucciones que se encuentran dentro de `{...}`. Si la condición no se cumple (es decir,

si su valor es `false`) no se ejecuta ninguna instrucción contenida en `{...}` y el programa continúa ejecutando el resto de instrucciones del script.

Ejemplo:

```
var mostrarMensaje = true;

if(mostrarMensaje) {
  alert("Hola Mundo");
}
```

En el ejemplo anterior, el mensaje sí que se muestra al usuario ya que la variable `mostrarMensaje` tiene un valor de `true` y por tanto, el programa entra dentro del bloque de instrucciones del `if`.

El ejemplo se podría reescribir también como:

```
var mostrarMensaje = true;

if(mostrarMensaje == true) {
  alert("Hola Mundo");
}
```

En este caso, la condición es una comparación entre el valor de la variable `mostrarMensaje` y el valor `true`. Como los dos valores coinciden, la igualdad se cumple y por tanto la condición es cierta, su valor es `true` y se ejecutan las instrucciones contenidas en ese bloque del `if`.

La comparación del ejemplo anterior suele ser el origen de muchos errores de programación, al confundir los operadores == y =. Las comparaciones siempre se realizan con el operador ==, ya que el operador = solamente asigna valores:

```
var mostrarMensaje = true;

// Se comparan los dos valores
if(mostrarMensaje == false) {
  ...
}

// Error - Se asigna el valor "false" a la variable
if(mostrarMensaje = false) {
  ...
}
```

La condición que controla el `if()` puede combinar los diferentes operadores lógicos y relacionales mostrados anteriormente:

```
var mostrado = false;

if(!mostrado) {
  alert("Es la primera vez que se muestra el mensaje");
}
```

Los operadores AND y OR permiten encadenar varias condiciones simples para construir condiciones complejas:

```
var mostrado = false;
var usuarioPermiteMensajes = true;

if(!mostrado && usuarioPermiteMensajes) {
  alert("Es la primera vez que se muestra el mensaje");
}
```

La condición anterior está formada por una operación AND sobre dos variables. A su vez, a la primera variable se le aplica el operador de negación antes de realizar la operación AND. De esta forma, como el valor de `mostrado` es `false`, el valor `!mostrado` sería `true`. Como la variable `usuarioPermiteMensajes` vale `true`, el resultado de `!mostrado && usuarioPermiteMensajes` sería igual a `true && true`, por lo que el resultado final de la condición del `if()` sería `true` y por tanto, se ejecutan las instrucciones que se encuentran dentro del bloque del `if()`.

Ejercicio 5

Completar las condiciones de los `if` del siguiente script para que los mensajes de los `alert()` se muestren siempre de forma correcta:

```
var numero1 = 5;
var numero2 = 8;

if(...) {
  alert("numero1 no es mayor que numero2");
}
if(...) {
  alert("numero2 es positivo");
}
if(...) {
  alert("numero1 es negativo o distinto de cero");
}
if(...) {
  alert("Incrementar en 1 unidad el valor de numero1 no lo hace mayor o igual
que numero2");
}
```

3.4.2. Estructura if...else

En ocasiones, las decisiones que se deben realizar no son del tipo *"si se cumple la condición, hazlo; si no se cumple, no hagas nada"*. Normalmente las condiciones suelen ser del tipo *"si se cumple esta condición, hazlo; si no se cumple, haz esto otro"*.

Para este segundo tipo de decisiones, existe una variante de la estructura `if` llamada `if...else`. Su definición formal es la siguiente:

```
if(condicion) {
  ...
}
else {
```

```
  ...
}
```

Si la condición se cumple (es decir, si su valor es `true`) se ejecutan todas las instrucciones que se encuentran dentro del `if()`. Si la condición no se cumple (es decir, si su valor es `false`) se ejecutan todas las instrucciones contenidas en `else { }`. Ejemplo:

```
var edad = 18;

if(edad >= 18) {
  alert("Eres mayor de edad");
}
else {
  alert("Todavía eres menor de edad");
}
```

Si el valor de la variable `edad` es mayor o igual que el valor numérico `18`, la condición del `if()` se cumple y por tanto, se ejecutan sus instrucciones y se muestra el mensaje `"Eres mayor de edad"`. Sin embargo, cuando el valor de la variable `edad` no es igual o mayor que `18`, la condición del `if()` no se cumple, por lo que automáticamente se ejecutan todas las instrucciones del bloque `else { }`. En este caso, se mostraría el mensaje `"Todavía eres menor de edad"`.

El siguiente ejemplo compara variables de tipo cadena de texto:

```
var nombre = "";

if(nombre == "") {
  alert("Aún no nos has dicho tu nombre");
}
else {
  alert("Hemos guardado tu nombre");
}
```

La condición del `if()` anterior se construye mediante el operador `==`, que es el que se emplea para comparar dos valores (no confundir con el operador `=` que se utiliza para asignar valores). En el ejemplo anterior, si la cadena de texto almacenada en la variable `nombre` es vacía (es decir, es igual a `""`) se muestra el mensaje definido en el `if()`. En otro caso, se muestra el mensaje definido en el bloque `else { }`.

La estructura `if...else` se puede encadenar para realizar varias comprobaciones seguidas:

```
if(edad < 12) {
  alert("Todavía eres muy pequeño");
}
else if(edad < 19) {
  alert("Eres un adolescente");
}
else if(edad < 35) {
  alert("Aun sigues siendo joven");
}
```

```
else {
  alert("Piensa en cuidarte un poco más");
}
```

No es obligatorio que la combinación de estructuras `if...else` acabe con la instrucción `else`, ya que puede terminar con una instrucción de tipo `else if()`.

Ejercicio 6

El cálculo de la letra del Documento Nacional de Identidad (DNI) es un proceso matemático sencillo que se basa en obtener el resto de la división entera del número de DNI y el número 23. A partir del resto de la división, se obtiene la letra seleccionándola dentro de un array de letras.

El array de letras es:

```
var letras = ['T', 'R', 'W', 'A', 'G', 'M', 'Y', 'F', 'P', 'D', 'X', 'B',
'N', 'J', 'Z', 'S', 'Q', 'V', 'H', 'L', 'C', 'K', 'E', 'T'];
```

Por tanto si el resto de la división es `0`, la letra del DNI es la `T` y si el resto es `3` la letra es la `A`. Con estos datos, elaborar un pequeño script que:

1. Almacene en una variable el número de DNI indicado por el usuario y en otra variable la letra del DNI que se ha indicado. (Pista: si se quiere pedir directamente al usuario que indique su número y su letra, se puede utilizar la función `prompt()`)
2. En primer lugar (y en una sola instrucción) se debe comprobar si el número es menor que `0` o mayor que `99999999`. Si ese es el caso, se muestra un mensaje al usuario indicando que el número proporcionado no es válido y el programa no muestra más mensajes.
3. Si el número es válido, se calcula la letra que le corresponde según el método explicado anteriormente.
4. Una vez calculada la letra, se debe comparar con la letra indicada por el usuario. Si no coinciden, se muestra un mensaje al usuario diciéndole que la letra que ha indicado no es correcta. En otro caso, se muestra un mensaje indicando que el número y la letra de DNI son correctos.

3.4.3. Estructura for

Las estructuras `if` y `if...else` no son muy eficientes cuando se desea ejecutar de forma repetitiva una instrucción. Por ejemplo, si se quiere mostrar un mensaje cinco veces, se podría pensar en utilizar el siguiente `if`:

```
var veces = 0;

if(veces < 4) {
  alert("Mensaje");
  veces++;
}
```

Se comprueba si la variable `veces` es menor que `4`. Si se cumple, se entra dentro del `if()`, se muestra el mensaje y se incrementa el valor de la variable `veces`. Así se debería seguir ejecutando hasta mostrar el mensaje las cinco veces deseadas.

Sin embargo, el funcionamiento real del script anterior es muy diferente al deseado, ya que solamente se muestra una vez el mensaje por pantalla. La razón es que la ejecución de la estructura `if()` no se repite y la comprobación de la condición sólo se realiza una vez, independientemente de que dentro del `if()` se modifique el valor de la variable utilizada en la condición.

La estructura `for` permite realizar este tipo de repeticiones (también llamadas bucles) de una forma muy sencilla. No obstante, su definición formal no es tan sencilla como la de `if()`:

```
for(inicializacion; condicion; actualizacion) {
  ...
}
```

La idea del funcionamiento de un bucle `for` es la siguiente: *"mientras la condición indicada se siga cumpliendo, repite la ejecución de las instrucciones definidas dentro del for. Además, después de cada repetición, actualiza el valor de las variables que se utilizan en la condición"*.

- La "inicialización" es la zona en la que se establece los valores iniciales de las variables que controlan la repetición.
- La "condición" es el único elemento que decide si continua o se detiene la repetición.
- La "actualización" es el nuevo valor que se asigna después de cada repetición a las variables que controlan la repetición.

```
var mensaje = "Hola, estoy dentro de un bucle";

for(var i = 0; i < 5; i++) {
  alert(mensaje);
}
```

La parte de la inicialización del bucle consiste en:

```
var i = 0;
```

Por tanto, en primer lugar se crea la variable `i` y se le asigna el valor de `0`. Esta zona de inicialización solamente se tiene en consideración justo antes de comenzar a ejecutar el bucle. Las siguientes repeticiones no tienen en cuenta esta parte de inicialización.

La zona de condición del bucle es:

```
i < 5
```

Los bucles se siguen ejecutando mientras se cumplan las condiciones y se dejan de ejecutar justo después de comprobar que la condición no se cumple. En este caso, mientras la variable `i` valga menos de `5` el bucle se ejecuta indefinidamente.

Como la variable i se ha inicializado a un valor de 0 y la condición para salir del bucle es que i sea menor que 5, si no se modifica el valor de i de alguna forma, el bucle se repetiría indefinidamente.

Por ese motivo, es imprescindible indicar la zona de actualización, en la que se modifica el valor de las variables que controlan el bucle:

```
i++
```

En este caso, el valor de la variable i se incrementa en una unidad después de cada repetición. La zona de actualización se ejecuta después de la ejecución de las instrucciones que incluye el for.

Así, durante la ejecución de la quinta repetición el valor de i será 4. Después de la quinta ejecución, se actualiza el valor de i, que ahora valdrá 5. Como la condición es que i sea menor que 5, la condición ya no se cumple y las instrucciones del for no se ejecutan una sexta vez.

Normalmente, la variable que controla los bucles for se llama i, ya que recuerda a la palabra índice y su nombre tan corto ahorra mucho tiempo y espacio.

El ejemplo anterior que mostraba los días de la semana contenidos en un array se puede rehacer de forma más sencilla utilizando la estructura for:

```
var dias = ["Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado",
"Domingo"];

for(var i=0; i<7; i++) {
  alert(dias[i]);
}
```

Ejercicio 7

El factorial de un número entero n es una operación matemática que consiste en multiplicar todos los factores `n x (n-1) x (n-2) x ... x 1`. Así, el factorial de 5 (escrito como `5!`) es igual a: `5! = 5 x 4 x 3 x 2 x 1 = 120`

Utilizando la estructura for, crear un script que calcule el factorial de un número entero.

3.4.4. Estructura for...in

Una estructura de control derivada de for es la estructura for...in. Su definición exacta implica el uso de objetos, que es un elemento de programación avanzada que no se va a estudiar. Por tanto, solamente se va a presentar la estructura for...in adaptada a su uso en arrays. Su definición formal adaptada a los arrays es:

```
for(indice in array) {
  ...
}
```

Si se quieren recorrer todos los elementos que forman un array, la estructura for...in es la forma más eficiente de hacerlo, como se muestra en el siguiente ejemplo:

```
var dias = ["Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado",
"Domingo"];

for(i in dias) {
  alert(dias[i]);
}
```

La variable que se indica como indice es la que se puede utilizar dentro del bucle `for...in` para acceder a los elementos del array. De esta forma, en la primera repetición del bucle la variable `i` vale `0` y en la última vale `6`.

Esta estructura de control es la más adecuada para recorrer arrays (y objetos), ya que evita tener que indicar la inicialización y las condiciones del bucle `for` simple y funciona correctamente cualquiera que sea la longitud del array. De hecho, sigue funcionando igual aunque varíe el número de elementos del array.

3.5. Funciones y propiedades básicas de JavaScript

JavaScript incorpora una serie de herramientas y utilidades (llamadas funciones y propiedades, como se verá más adelante) para el manejo de las variables. De esta forma, muchas de las operaciones básicas con las variables, se pueden realizar directamente con las utilidades que ofrece JavaScript.

3.5.1. Funciones útiles para cadenas de texto

A continuación se muestran algunas de las funciones más útiles para el manejo de cadenas de texto:

`length`, calcula la longitud de una cadena de texto (el número de caracteres que la forman)

```
var mensaje = "Hola Mundo";
var numeroLetras = mensaje.length; // numeroLetras = 10
```

`+`, se emplea para concatenar varias cadenas de texto

```
var mensaje1 = "Hola";
var mensaje2 = " Mundo";
var mensaje = mensaje1 + mensaje2; // mensaje = "Hola Mundo"
```

Además del operador `+`, también se puede utilizar la función `concat()`

```
var mensaje1 = "Hola";
var mensaje2 = mensaje1.concat(" Mundo"); // mensaje2 = "Hola Mundo"
```

Las cadenas de texto también se pueden unir con variables numéricas:

```
var variable1 = "Hola ";
var variable2 = 3;
var mensaje = variable1 + variable2; // mensaje = "Hola 3"
```

Cuando se unen varias cadenas de texto es habitual olvidar añadir un espacio de separación entre las palabras:

```
var mensaje1 = "Hola";
var mensaje2 = "Mundo";
var mensaje = mensaje1 + mensaje2; // mensaje = "HolaMundo"
```

Los espacios en blanco se pueden añadir al final o al principio de las cadenas y también se pueden indicar forma explícita:

```
var mensaje1 = "Hola";
var mensaje2 = "Mundo";
var mensaje = mensaje1 + " " + mensaje2; // mensaje = "Hola Mundo"
```

`toUpperCase()`, transforma todos los caracteres de la cadena a sus correspondientes caracteres en mayúsculas:

```
var mensaje1 = "Hola";
var mensaje2 = mensaje1.toUpperCase(); // mensaje2 = "HOLA"
```

`toLowerCase()`, transforma todos los caracteres de la cadena a sus correspondientes caracteres en minúsculas:

```
var mensaje1 = "HolA";
var mensaje2 = mensaje1.toLowerCase(); // mensaje2 = "hola"
```

`charAt(posicion)`, obtiene el carácter que se encuentra en la posición indicada:

```
var mensaje = "Hola";
var letra = mensaje.charAt(0); // letra = H
letra = mensaje.charAt(2);     // letra = l
```

`indexOf(caracter)`, calcula la posición en la que se encuentra el carácter indicado dentro de la cadena de texto. Si el carácter se incluye varias veces dentro de la cadena de texto, se devuelve su primera posición empezando a buscar desde la izquierda. Si la cadena no contiene el carácter, la función devuelve el valor `-1`:

```
var mensaje = "Hola";
var posicion = mensaje.indexOf('a'); // posicion = 3
posicion = mensaje.indexOf('b');     // posicion = -1
```

Su función análoga es `lastIndexOf()`:

`lastIndexOf(caracter)`, calcula la última posición en la que se encuentra el carácter indicado dentro de la cadena de texto. Si la cadena no contiene el carácter, la función devuelve el valor `-1`:

```
var mensaje = "Hola";
var posicion = mensaje.lastIndexOf('a'); // posicion = 3
posicion = mensaje.lastIndexOf('b');     // posicion = -1
```

La función `lastIndexOf()` comienza su búsqueda desde el final de la cadena hacia el principio, aunque la posición devuelta es la correcta empezando a contar desde el principio de la palabra.

`substring(inicio, final)`, extrae una porción de una cadena de texto. El segundo parámetro es opcional. Si sólo se indica el parámetro `inicio`, la función devuelve la parte de la cadena original correspondiente desde esa posición hasta el final:

```
var mensaje = "Hola Mundo";
var porcion = mensaje.substring(2); // porcion = "la Mundo"
porcion = mensaje.substring(5);     // porcion = "Mundo"
porcion = mensaje.substring(7);     // porcion = "ndo"
```

Si se indica un `inicio` negativo, se devuelve la misma cadena original:

```
var mensaje = "Hola Mundo";
var porcion = mensaje.substring(-2); // porcion = "Hola Mundo"
```

Cuando se indica el inicio y el final, se devuelve la parte de la cadena original comprendida entre la posición inicial y la inmediatamente anterior a la posición final (es decir, la posición `inicio` está incluida y la posición `final` no):

```
var mensaje = "Hola Mundo";
var porcion = mensaje.substring(1, 8); // porcion = "ola Mun"
porcion = mensaje.substring(3, 4);     // porcion = "a"
```

Si se indica un `final` más pequeño que el `inicio`, JavaScript los considera de forma inversa, ya que automáticamente asigna el valor más pequeño al `inicio` y el más grande al `final`:

```
var mensaje = "Hola Mundo";
var porcion = mensaje.substring(5, 0); // porcion = "Hola "
porcion = mensaje.substring(0, 5);     // porcion = "Hola "
```

`split(separador)`, convierte una cadena de texto en un array de cadenas de texto. La función parte la cadena de texto determinando sus trozos a partir del carácter `separador` indicado:

```
var mensaje = "Hola Mundo, soy una cadena de texto!";
var palabras = mensaje.split(" ");
// palabras = ["Hola", "Mundo,", "soy", "una", "cadena", "de", "texto!"];
```

Con esta función se pueden extraer fácilmente las letras que forman una palabra:

```
var palabra = "Hola";
var letras = palabra.split(""); // letras = ["H", "o", "l", "a"]
```

3.5.2. Funciones útiles para arrays

A continuación se muestran algunas de las funciones más útiles para el manejo de arrays:

`length`, calcula el número de elementos de un array

```
var vocales = ["a", "e", "i", "o", "u"];
var numeroVocales = vocales.length; // numeroVocales = 5
```

`concat()`, se emplea para concatenar los elementos de varios arrays

```
var array1 = [1, 2, 3];
array2 = array1.concat(4, 5, 6);   // array2 = [1, 2, 3, 4, 5, 6]
array3 = array1.concat([4, 5, 6]); // array3 = [1, 2, 3, 4, 5, 6]
```

`join(separador)`, es la función contraria a `split()`. Une todos los elementos de un array para formar una cadena de texto. Para unir los elementos se utiliza el carácter `separador` indicado

```
var array = ["hola", "mundo"];
var mensaje = array.join(""); // mensaje = "holamundo"
mensaje = array.join(" ");    // mensaje = "hola mundo"
```

`pop()`, elimina el último elemento del array y lo devuelve. El array original se modifica y su longitud disminuye en 1 elemento.

```
var array = [1, 2, 3];
var ultimo = array.pop();
// ahora array = [1, 2], ultimo = 3
```

`push()`, añade un elemento al final del array. El array original se modifica y aumenta su longitud en 1 elemento. (También es posible añadir más de un elemento a la vez)

```
var array = [1, 2, 3];
array.push(4);
// ahora array = [1, 2, 3, 4]
```

`shift()`, elimina el primer elemento del array y lo devuelve. El array original se ve modificado y su longitud disminuida en 1 elemento.

```
var array = [1, 2, 3];
var primero = array.shift();
// ahora array = [2, 3], primero = 1
```

`unshift()`, añade un elemento al principio del array. El array original se modifica y aumenta su longitud en 1 elemento. (También es posible añadir más de un elemento a la vez)

```
var array = [1, 2, 3];
array.unshift(0);
// ahora array = [0, 1, 2, 3]
```

`reverse()`, modifica un array colocando sus elementos en el orden inverso a su posición original:

```
var array = [1, 2, 3];
array.reverse();
// ahora array = [3, 2, 1]
```

3.5.3. Funciones útiles para números

A continuación se muestran algunas de las funciones y propiedades más útiles para el manejo de números.

`NaN`, (del inglés, *"Not a Number"*) JavaScript emplea el valor `NaN` para indicar un valor numérico no definido (por ejemplo, la división `0/0`).

```
var numero1 = 0;
var numero2 = 0;
alert(numero1/numero2); // se muestra el valor NaN
```

`isNaN()`, permite proteger a la aplicación de posibles valores numéricos no definidos

```
var numero1 = 0;
var numero2 = 0;
if(isNaN(numero1/numero2)) {
  alert("La división no está definida para los números indicados");
}
else {
  alert("La división es igual a => " + numero1/numero2);
}
```

`Infinity`, hace referencia a un valor numérico infinito y positivo (también existe el valor `–Infinity` para los infinitos negativos)

```
var numero1 = 10;
var numero2 = 0;
alert(numero1/numero2); // se muestra el valor Infinity
```

`toFixed(digitos)`, devuelve el número original con tantos decimales como los indicados por el parámetro `digitos` y realiza los redondeos necesarios. Se trata de una función muy útil por ejemplo para mostrar precios.

```
var numero1 = 4564.34567;
numero1.toFixed(2); // 4564.35
numero1.toFixed(6); // 4564.345670
numero1.toFixed(); // 4564
```

Capítulo 4. Programación avanzada

Las estructuras de control, los operadores y todas las utilidades propias de JavaScript que se han visto en los capítulos anteriores, permiten crear scripts sencillos y de mediana complejidad.

Sin embargo, para las aplicaciones más complejas son necesarios otros elementos como las funciones y otras estructuras de control más avanzadas, que se describen en este capítulo.

4.1. Funciones

Cuando se desarrolla una aplicación compleja, es muy habitual utilizar una y otra vez las mismas instrucciones. Un script para una tienda de comercio electrónico por ejemplo, tiene que calcular el precio total de los productos varias veces, para añadir los impuestos y los gastos de envío.

Cuando una serie de instrucciones se repiten una y otra vez, se complica demasiado el código fuente de la aplicación, ya que:

- El código de la aplicación es mucho más largo porque muchas instrucciones están repetidas.
- Si se quiere modificar alguna de las instrucciones repetidas, se deben hacer tantas modificaciones como veces se haya escrito esa instrucción, lo que se convierte en un trabajo muy pesado y muy propenso a cometer errores.

Las funciones son la solución a todos estos problemas, tanto en JavaScript como en el resto de lenguajes de programación. Una función es un conjunto de instrucciones que se agrupan para realizar una tarea concreta y que se pueden reutilizar fácilmente.

En el siguiente ejemplo, las instrucciones que suman los dos números y muestran un mensaje con el resultado se repiten una y otra vez:

```
var resultado;

var numero1 = 3;
var numero2 = 5;

// Se suman los números y se muestra el resultado
resultado = numero1 + numero2;
alert("El resultado es " + resultado);

numero1 = 10;
numero2 = 7;

// Se suman los números y se muestra el resultado
resultado = numero1 + numero2;
alert("El resultado es " + resultado);
```

```
numero1 = 5;
numero2 = 8;

// Se suman los números y se muestra el resultado
resultado = numero1 + numero2;
alert("El resultado es " + resultado);
...
```

Aunque es un ejemplo muy sencillo, parece evidente que repetir las mismas instrucciones a lo largo de todo el código no es algo recomendable. La solución que proponen las funciones consiste en extraer las instrucciones que se repiten y sustituirlas por una instrucción del tipo *"en este punto, se ejecutan las instrucciones que se han extraído"*:

```
var resultado;

var numero1 = 3;
var numero2 = 5;

/* En este punto, se llama a la función que suma
    2 números y muestra el resultado */

numero1 = 10;
numero2 = 7;

/* En este punto, se llama a la función que suma
    2 números y muestra el resultado */

numero1 = 5;
numero2 = 8;

/* En este punto, se llama a la función que suma
    2 números y muestra el resultado */
...
```

Para que la solución del ejemplo anterior sea válida, las instrucciones comunes se tienen que agrupar en una función a la que se le puedan indicar los números que debe sumar antes de mostrar el mensaje.

Por lo tanto, en primer lugar se debe crear la función básica con las instrucciones comunes. Las funciones en JavaScript se definen mediante la palabra reservada `function`, seguida del nombre de la función. Su definición formal es la siguiente:

```
function nombre_funcion() {
  ...
}
```

El nombre de la función se utiliza para *llamar* a esa función cuando sea necesario. El concepto es el mismo que con las variables, a las que se les asigna un nombre único para poder utilizarlas dentro del código. Después del nombre de la función, se incluyen dos paréntesis cuyo significado se detalla más adelante. Por último, los símbolos `{` y `}` se utilizan para encerrar todas las instrucciones que pertenecen a la función (de forma similar a como se encierran las instrucciones en las estructuras `if` o `for`).

Volviendo al ejemplo anterior, se crea una función llamada `suma_y_muestra` de la siguiente forma:

```
function suma_y_muestra() {
  resultado = numero1 + numero2;
  alert("El resultado es " + resultado);
}
```

Aunque la función anterior está correctamente creada, no funciona como debería ya que le faltan los *"argumentos"*, que se explican en la siguiente sección. Una vez creada la función, desde cualquier punto del código se puede *llamar* a la función para que se ejecuten sus instrucciones (además de *"llamar a la función"*, también se suele utilizar la expresión *"invocar a la función"*).

La llamada a la función se realiza simplemente indicando su nombre, incluyendo los paréntesis del final y el carácter `;` para terminar la instrucción:

```
function suma_y_muestra() {
  resultado = numero1 + numero2;
  alert("El resultado es " + resultado);
}

var resultado;

var numero1 = 3;
var numero2 = 5;

suma_y_muestra();

numero1 = 10;
numero2 = 7;

suma_y_muestra();

numero1 = 5;
numero2 = 8;

suma_y_muestra();
...
```

El código del ejemplo anterior es mucho más eficiente que el primer código que se mostró, ya que no existen instrucciones repetidas. Las instrucciones que suman y muestran mensajes se han agrupado bajo una función, lo que permite ejecutarlas en cualquier punto del programa simplemente indicando el nombre de la función.

Lo único que le falta al ejemplo anterior para funcionar correctamente es poder indicar a la función los números que debe sumar. Cuando se necesitan pasar datos a una función, se utilizan los *"argumentos"*, como se explica en la siguiente sección.

4.1.1. Argumentos y valores de retorno

Las funciones más sencillas no necesitan ninguna información para producir sus resultados. Sin embargo, la mayoría de funciones de las aplicaciones reales deben acceder al valor de algunas variables para producir sus resultados.

Las variables que necesitan las funciones se llaman *argumentos*. Antes de que pueda utilizarlos, la función debe indicar cuántos argumentos necesita y cuál es el nombre de cada argumento. Además, al invocar la función, se deben incluir los valores que se le van a pasar a la función. Los argumentos se indican dentro de los paréntesis que van detrás del nombre de la función y se separan con una coma (`,`).

Siguiendo el ejemplo anterior, la función debe indicar que necesita dos argumentos, correspondientes a los dos números que tiene que sumar:

```
function suma_y_muestra(primerNumero, segundoNumero) { ... }
```

A continuación, para utilizar el valor de los argumentos dentro de la función, se debe emplear el mismo nombre con el que se definieron los argumentos:

```
function suma_y_muestra(primerNumero, segundoNumero) { ... }
  var resultado = primerNumero + segundoNumero;
  alert("El resultado es " + resultado);
}
```

Dentro de la función, el valor de la variable `primerNumero` será igual al primer valor que se le pase a la función y el valor de la variable `segundoNumero` será igual al segundo valor que se le pasa. Para pasar valores a la función, se incluyen dentro de los paréntesis utilizados al llamar a la función:

```
// Definición de la función
function suma_y_muestra(primerNumero, segundoNumero) { ... }
  var resultado = primerNumero + segundoNumero;
  alert("El resultado es " + resultado);
}

// Declaración de las variables
var numero1 = 3;
var numero2 = 5;

// Llamada a la función
suma_y_muestra(numero1, numero2);
```

En el código anterior, se debe tener en cuenta que:

- Aunque casi siempre se utilizan variables para pasar los datos a la función, se podría haber utilizado directamente el valor de esas variables: `suma_y_muestra(3, 5);`
- El número de argumentos que se pasa a una función debería ser el mismo que el número de argumentos que ha indicado la función. No obstante, JavaScript no muestra ningún error si se pasan más o menos argumentos de los necesarios.

- El orden de los argumentos es fundamental, ya que el primer dato que se indica en la llamada, será el primer valor que espera la función; el segundo valor indicado en la llamada, es el segundo valor que espera la función y así sucesivamente.
- Se puede utilizar un número ilimitado de argumentos, aunque si su número es muy grande, se complica en exceso la llamada a la función.
- No es obligatorio que coincida el nombre de los argumentos que utiliza la función y el nombre de los argumentos que se le pasan. En el ejemplo anterior, los argumentos que se pasan son `numero1` y `numero2` y los argumentos que utiliza la función son `primerNumero` y `segundoNumero`.

A continuación se muestra otro ejemplo de una función que calcula el precio total de un producto a partir de su precio básico:

```
// Definición de la función
function calculaPrecioTotal(precio) {
  var impuestos = 1.16;
  var gastosEnvio = 10;
  var precioTotal = ( precio * impuestos ) + gastosEnvio;
}

// Llamada a la función
calculaPrecioTotal(23.34);
```

La función anterior toma como argumento una variable llamada `precio` y le suma los impuestos y los gastos de envío para obtener el precio total. Al llamar a la función, se pasa directamente el valor del precio básico mediante el número `23.34`.

No obstante, el código anterior no es demasiado útil, ya que lo ideal sería que la función pudiera devolver el resultado obtenido para guardarlo en otra variable y poder seguir trabajando con este precio total:

```
function calculaPrecioTotal(precio) {
  var impuestos = 1.16;
  var gastosEnvio = 10;
  var precioTotal = ( precio * impuestos ) + gastosEnvio;
}

// El valor devuelto por la función, se guarda en una variable
var precioTotal = calculaPrecioTotal(23.34);

// Seguir trabajando con la variable "precioTotal"
```

Afortunadamente, las funciones no solamente puede recibir variables y datos, sino que también pueden devolver los valores que han calculado. Para devolver valores dentro de una función, se utiliza la palabra reservada `return`. Aunque las funciones pueden devolver valores de cualquier tipo, solamente pueden devolver un valor cada vez que se ejecutan.

```
function calculaPrecioTotal(precio) {
  var impuestos = 1.16;
  var gastosEnvio = 10;
```

```
  var precioTotal = ( precio * impuestos ) + gastosEnvio;
  return precioTotal;
}

var precioTotal = calculaPrecioTotal(23.34);

// Seguir trabajando con la variable "precioTotal"
```

Para que la función devuelva un valor, solamente es necesario escribir la palabra reservada `return` junto con el nombre de la variable que se quiere devolver. En el ejemplo anterior, la ejecución de la función llega a la instrucción `return precioTotal;` y en ese momento, devuelve el valor que contenga la variable `precioTotal`.

Como la función devuelve un valor, en el punto en el que se realiza la llamada, debe indicarse el nombre de una variable en el que se guarda el valor devuelto:

```
var precioTotal = calculaPrecioTotal(23.34);
```

Si no se indica el nombre de ninguna variable, JavaScript no muestra ningún error y el valor devuelto por la función simplemente se pierde y por tanto, no se utilizará en el resto del programa. En este caso, tampoco es obligatorio que el nombre de la variable devuelta por la función coincida con el nombre de la variable en la que se va a almacenar ese valor.

Si la función llega a una instrucción de tipo `return`, se devuelve el valor indicado y finaliza la ejecución de la función. Por tanto, todas las instrucciones que se incluyen después de un `return` se ignoran y por ese motivo la instrucción `return` suele ser la última de la mayoría de funciones.

Para que el ejemplo anterior sea más completo, se puede añadir otro argumento a la función que indique el porcentaje de impuestos que se debe añadir al precio del producto. Evidentemente, el nuevo argumento se debe añadir tanto a la definición de la función como a su llamada:

```
function calculaPrecioTotal(precio, porcentajeImpuestos) {
  var gastosEnvio = 10;
  var precioConImpuestos = (1 + porcentajeImpuestos/100) * precio;
  var precioTotal = precioConImpuestos + gastosEnvio;
  return precioTotal;
}

var precioTotal = calculaPrecioTotal(23.34, 16);
var otroPrecioTotal = calculaPrecioTotal(15.20, 4);
```

Para terminar de completar el ejercicio anterior, se puede rendondear a dos decimales el precio total devuelto por la función:

```
function calculaPrecioTotal(precio, porcentajeImpuestos) {
  var gastosEnvio = 10;
  var precioConImpuestos = (1 + porcentajeImpuestos/100) * precio;
  var precioTotal = precioConImpuestos + gastosEnvio;
  return precioTotal.toFixed(2);
}
```

```
var precioTotal = calculaPrecioTotal(23.34, 16);
```

Ejercicio 8

Escribir el código de una función a la que se pasa como parámetro un número entero y devuelve como resultado una cadena de texto que indica si el número es par o impar. Mostrar por pantalla el resultado devuelto por la función.

Ejercicio 9

Definir una función que muestre información sobre una cadena de texto que se le pasa como argumento. A partir de la cadena que se le pasa, la función determina si esa cadena está formada sólo por mayúsculas, sólo por minúsculas o por una mezcla de ambas.

Ejercicio 10

Definir una función que determine si la cadena de texto que se le pasa como parámetro es un palíndromo, es decir, si se lee de la misma forma desde la izquierda y desde la derecha. Ejemplo de palíndromo complejo: "La ruta nos aporto otro paso natural".

4.2. Ámbito de las variables

El ámbito de una variable (llamado *"scope"* en inglés) es la zona del programa en la que se define la variable. JavaScript define dos ámbitos para las variables: global y local.

El siguiente ejemplo ilustra el comportamiento de los ámbitos:

```
function creaMensaje() {
  var mensaje = “Mensaje de prueba”;
}
creaMensaje();
alert(mensaje);
```

El ejemplo anterior define en primer lugar una función llamada `creaMensaje` que crea una variable llamada `mensaje`. A continuación, se ejecuta la función mediante la llamada `creaMensaje();` y seguidamente, se muestra mediante la función `alert()` el valor de una variable llamada `mensaje`.

Sin embargo, al ejecutar el código anterior no se muestra ningún mensaje por pantalla. La razón es que la variable `mensaje` se ha definido dentro de la función `creaMensaje()` y por tanto, es una **variable local** que solamente está definida dentro de la función.

Cualquier instrucción que se encuentre dentro de la función puede hacer uso de esa variable, pero todas las instrucciones que se encuentren en otras funciones o fuera de cualquier función no tendrán definida la variable `mensaje`. De esta forma, para mostrar el mensaje en el código anterior, la función `alert()` debe llamarse desde dentro de la función `creaMensaje()`:

```
function creaMensaje() {
  var mensaje = “Mensaje de prueba”;
  alert(mensaje);
```

```
}
creaMensaje();
```

Además de variables locales, también existe el concepto de **variable global**, que está definida en cualquier punto del programa (incluso dentro de cualquier función).

```
var mensaje = "Mensaje de prueba";

function muestraMensaje() {
  alert(mensaje);
}
```

El código anterior es el ejemplo inverso al mostrado anteriormente. Dentro de la función `muestraMensaje()` se quiere hacer uso de una variable llamada `mensaje` y que no ha sido definida dentro de la propia función. Sin embargo, si se ejecuta el código anterior, sí que se muestra el mensaje definido por la variable `mensaje`.

El motivo es que en el código JavaScript anterior, la variable `mensaje` se ha definido fuera de cualquier función. Este tipo de variables automáticamente se transforman en variables globales y están disponibles en cualquier punto del programa (incluso dentro de cualquier función).

De esta forma, aunque en el interior de la función no se ha definido ninguna variable llamada `mensaje`, la variable global creada anteriormente permite que la instrucción `alert()` dentro de la función muestre el mensaje correctamente.

Si una variable se declara fuera de cualquier función, automáticamente se transforma en variable global independientemente de si se define utilizando la palabra reservada `var` o no. Sin embargo, las variables definidas dentro de una función pueden ser globales o locales.

Si en el interior de una función, las variables se declaran mediante `var` se consideran locales y las variables que no se han declarado mediante `var`, se transforman automáticamente en variables globales.

Por lo tanto, se puede rehacer el código del primer ejemplo para que muestre el mensaje correctamente. Para ello, simplemente se debe definir la variable dentro de la función sin la palabra reservada `var`, para que se transforme en una variable global:

```
function creaMensaje() {
  mensaje = "Mensaje de prueba";
}

creaMensaje();
alert(mensaje);
```

¿Qué sucede si una función define una variable local con el mismo nombre que una variable global que ya existe? En este caso, las variables locales prevalecen sobre las globales, pero sólo dentro de la función:

```
var mensaje = "gana la de fuera";
```

```
function muestraMensaje() {
  var mensaje = "gana la de dentro";
  alert(mensaje);
}

alert(mensaje);
muestraMensaje();
alert(mensaje);
```

El código anterior muestra por pantalla los siguientes mensajes:

```
gana la de fuera
gana la de dentro
gana la de fuera
```

Dentro de la función, la variable local llamada `mensaje` tiene más prioridad que la variable global del mismo nombre, pero solamente dentro de la función.

¿Qué sucede si dentro de una función se define una variable global con el mismo nombre que otra variable global que ya existe? En este otro caso, la variable global definida dentro de la función simplemente modifica el valor de la variable global definida anteriormente:

```
var mensaje = "gana la de fuera";
function muestraMensaje() {
  mensaje = "gana la de dentro";
  alert(mensaje);
}

alert(mensaje);
muestraMensaje();
alert(mensaje);
```

En este caso, los mensajes mostrados son:

```
gana la de fuera
gana la de dentro
gana la de dentro
```

La recomendación general es definir como variables locales todas las variables que sean de uso exclusivo para realizar las tareas encargadas a cada función. Las variables globales se utilizan para compartir variables entre funciones de forma sencilla.

4.3. Sentencias break y continue

La estructura de control `for` es muy sencilla de utilizar, pero tiene el inconveniente de que el número de repeticiones que se realizan sólo se pueden controlar mediante las variables definidas en la zona de actualización del bucle.

Las sentencias `break` y `continue` permiten manipular el comportamiento normal de los bucles `for` para detener el bucle o para saltarse algunas repeticiones. Concretamente, la sentencia `break` permite terminar de forma abrupta un bucle y la sentencia `continue` permite saltarse algunas repeticiones del bucle.

El siguiente ejemplo muestra el uso de la sentencia `break`:

```
var cadena = "En un lugar de la Mancha de cuyo nombre no quiero acordarme...";
var letras = cadena.split("");
var resultado = "";

for(i in letras) {
  if(letras[i] == 'a') {
    break;
  }
  else {
    resultado += letras[i];
  }
}
alert(resultado);
// muestra "En un lug"
```

Si el programa llega a una instrucción de tipo `break;`, sale inmediatamente del bucle y continúa ejecutando el resto de instrucciones que se encuentran fuera del bucle `for`. En el ejemplo anterior, se recorren todas las letras de una cadena de texto y cuando se encuentra con la primera letra `"a"`, se detiene la ejecución del bucle `for`.

La utilidad de `break` es terminar la ejecución del bucle cuando una variable toma un determinado valor o cuando se cumple alguna condición.

En ocasiones, lo que se desea es saltarse alguna repetición del bucle cuando se dan algunas condiciones. Siguiendo con el ejemplo anterior, ahora se desea que el texto de salida elimine todas las letras `"a"` de la cadena de texto original:

```
var cadena = "En un lugar de la Mancha de cuyo nombre no quiero acordarme...";
var letras = cadena.split("");
var resultado = "";

for(i in letras) {
  if(letras[i] == 'a') {
    continue;
  }
  else {
    resultado += letras[i];
  }
}
alert(resultado);
// muestra "En un lugr de l Mnch de cuyo nombre no quiero cordrme..."
```

En este caso, cuando se encuentra una letra `"a"` no se termina el bucle, sino que no se ejecutan las instrucciones de esa repetición y se pasa directamente a la siguiente repetición del bucle `for`.

La utilidad de `continue` es que permite utilizar el bucle `for` para filtrar los resultados en función de algunas condiciones o cuando el valor de alguna variable coincide con un valor determinado.

4.4. Otras estructuras de control

Las estructuras de control de flujo que se han visto (`if`, `else`, `for`) y las sentencias que modifican su comportamiento (`break`, `continue`) no son suficientes para realizar algunas tareas complejas y otro tipo de repeticiones. Por ese motivo, JavaScript proporciona otras estructuras de control de flujo diferentes y en algunos casos más eficientes.

4.4.1. Estructura while

La estructura `while` permite crear bucles que se ejecutan ninguna o más veces, dependiendo de la condición indicada. Su definición formal es:

```
while(condicion) {
  ...
}
```

El funcionamiento del bucle `while` se resume en: *"mientras se cumpla la condición indicada, repite indefinidamente las instrucciones incluidas dentro del bucle"*.

Si la condición no se cumple ni siquiera la primera vez, el bucle no se ejecuta. Si la condición se cumple, se ejecutan las instrucciones una vez y se vuelve a comprobar la condición. Si se sigue cumpliendo la condición, se vuelve a ejecutar el bucle y así se continúa hasta que la condición no se cumpla.

Evidentemente, las variables que controlan la condición deben modificarse dentro del propio bucle, ya que de otra forma, la condición se cumpliría siempre y el bucle `while` se repetiría indefinidamente.

El siguiente ejemplo utiliza el bucle `while` para sumar todos los números menores o iguales que otro número:

```
var resultado = 0;
var numero = 100;
var i = 0;

while(i <= numero) {
  resultado += i;
  i++;
}

alert(resultado);
```

El programa debe sumar todos los números menores o igual que otro dado. Por ejemplo si el número es `5`, se debe calcular: `1 + 2 + 3 + 4 + 5 = 15`

Este tipo de condiciones (*"suma números mientras sean menores o iguales que otro número dado"*) se resuelven muy fácilmente con los bucles tipo `while`, aunque también se podían resolver con bucles de tipo `for`.

En el ejemplo anterior, mientras se cumpla la condición, es decir, mientras que la variable `i` sea menor o igual que la variable `numero`, se ejecutan las instrucciones del bucle.

Dentro del bucle se suma el valor de la variable `i` al resultado total (variable `resultado`) y se actualiza el valor de la variable `i`, que es la que controla la condición del bucle. Si no se actualiza el valor de la variable `i`, la ejecución del bucle continua infinitamente o hasta que el navegador permita al usuario detener el script.

4.4.2. Estructura do...while

El bucle de tipo `do...while` es muy similar al bucle `while`, salvo que en este caso **siempre** se ejecutan las instrucciones del bucle al menos la primera vez. Su definición formal es:

```
do {
  ...
} while(condicion);
```

De esta forma, como la condición se comprueba después de cada repetición, la primera vez siempre se ejecutan las instrucciones del bucle. Es importante no olvidar que después del `while()` se debe añadir el carácter `;` (al contrario de lo que sucede con el bucle `while` simple).

Utilizando este bucle se puede calcular fácilmente el factorial de un número:

```
var resultado = 1;
var numero = 5;

do {
  resultado *= numero;  // resultado = resultado * numero
  numero--;
} while(numero > 0);

alert(resultado);
```

En el código anterior, el `resultado` se multiplica en cada repetición por el valor de la variable `numero`. Además, en cada repetición se decrementa el valor de esta variable `numero`. La condición del bucle `do...while` es que el valor de `numero` sea mayor que `0`, ya que el factorial de un número multiplica todos los números menores o iguales que él mismo, pero hasta el número 1 (el factorial de 5 por ejemplo es `5 x 4 x 3 x 2 x 1 = 120`).

Como en cada repetición se decrementa el valor de la variable `numero` y la condición es que `numero` sea mayor que cero, en la repetición en la que `numero` valga `0`, la condición ya no se cumple y el programa se sale del bucle `do...while`.

4.4.3. Estructura switch

La estructura `if...else` se puede utilizar para realizar comprobaciones múltiples y tomar decisiones complejas. Sin embargo, si todas las condiciones dependen siempre de la misma variable, el código JavaScript resultante es demasiado redundante:

```
if(numero == 5) {
  ...
}
else if(numero == 8) {
  ...
}
else if(numero == 20) {
  ...
}
else {
  ...
}
```

En estos casos, la estructura `switch` es la más eficiente, ya que está especialmente diseñada para manejar de forma sencilla múltiples condiciones sobre la misma variable. Su definición formal puede parecer compleja, aunque su uso es muy sencillo:

```
switch(variable) {
  case valor_1:
    ...
    break;
  case valor_2:
    ...
    break;
  ...
  case valor_n:
    ...
    break;
  default:
    ...
    break;
}
```

El anterior ejemplo realizado con `if...else` se puede rehacer mediante `switch`:

```
switch(numero) {
  case 5:
    ...
    break;
  case 8:
    ...
    break;
  case 20:
    ...
    break;
  default:
    ...
    break;
}
```

La estructura `switch` se define mediante la palabra reservada `switch` seguida, entre paréntesis, del nombre de la variable que se va a utilizar en las comparaciones. Como es habitual, las instrucciones que forman parte del `switch` se encierran entre las llaves `{` y `}`.

Dentro del `switch` se definen todas las comparaciones que se quieren realizar sobre el valor de la variable. Cada comparación se indica mediante la palabra reservada `case` seguida del valor con el que se realiza la comparación. Si el valor de la variable utilizada por `switch` coincide con el valor indicado por `case`, se ejecutan las instrucciones definidas dentro de ese `case`.

Normalmente, después de las instrucciones de cada `case` se incluye la sentencia `break` para terminar la ejecución del `switch`, aunque no es obligatorio. Las comparaciones se realizan por orden, desde el primer `case` hasta el último, por lo que es muy importante el orden en el que se definen los `case`.

¿Qué sucede si ningún valor de la variable del `switch` coincide con los valores definidos en los `case`? En este caso, se utiliza el valor `default` para indicar las instrucciones que se ejecutan en el caso en el que ningún `case` se cumpla para la variable indicada.

Aunque `default` es opcional, las estructuras `switch` suelen incluirlo para definir al menos un valor por defecto para alguna variable o para mostrar algún mensaje por pantalla.

Capítulo 5. DOM

La creación del *Document Object Model* o **DOM** es una de las innovaciones que más ha influido en el desarrollo de las páginas web dinámicas y de las aplicaciones web más complejas.

DOM permite a los programadores web acceder y manipular las páginas XHTML como si fueran documentos XML. De hecho, DOM se diseñó originalmente para manipular de forma sencilla los documentos XML.

A pesar de sus orígenes, DOM se ha convertido en una utilidad disponible para la mayoría de lenguajes de programación (Java, PHP, JavaScript) y cuyas únicas diferencias se encuentran en la forma de implementarlo.

5.1. Árbol de nodos

Una de las tareas habituales en la programación de aplicaciones web con JavaScript consiste en la manipulación de las páginas web. De esta forma, es habitual obtener el valor almacenado por algunos elementos (por ejemplo los elementos de un formulario), crear un elemento (párrafos, `<div>`, etc.) de forma dinámica y añadirlo a la página, aplicar una animación a un elemento (que aparezca/desaparezca, que se desplace, etc.).

Todas estas tareas habituales son muy sencillas de realizar gracias a DOM. Sin embargo, para poder utilizar las utilidades de DOM, es necesario *"transformar"* la página original. Una página HTML normal no es más que una sucesión de caracteres, por lo que es un formato muy difícil de manipular. Por ello, los navegadores web transforman automáticamente todas las páginas web en una estructura más eficiente de manipular.

Esta transformación la realizan todos los navegadores de forma automática y nos permite utilizar las herramientas de DOM de forma muy sencilla. El motivo por el que se muestra el funcionamiento de esta transformación interna es que condiciona el comportamiento de DOM y por tanto, la forma en la que se manipulan las páginas.

DOM transforma todos los documentos XHTML en un conjunto de elementos llamados *nodos*, que están interconectados y que representan los contenidos de las páginas web y las relaciones entre ellos. Por su aspecto, la unión de todos los nodos se llama *"árbol de nodos"*.

La siguiente página XHTML sencilla:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Página sencilla</title>
</head>
```

```
<body>
<p>Esta página es <strong>muy sencilla</strong></p>
</body>
</html>
```

Se transforma en el siguiente árbol de nodos:

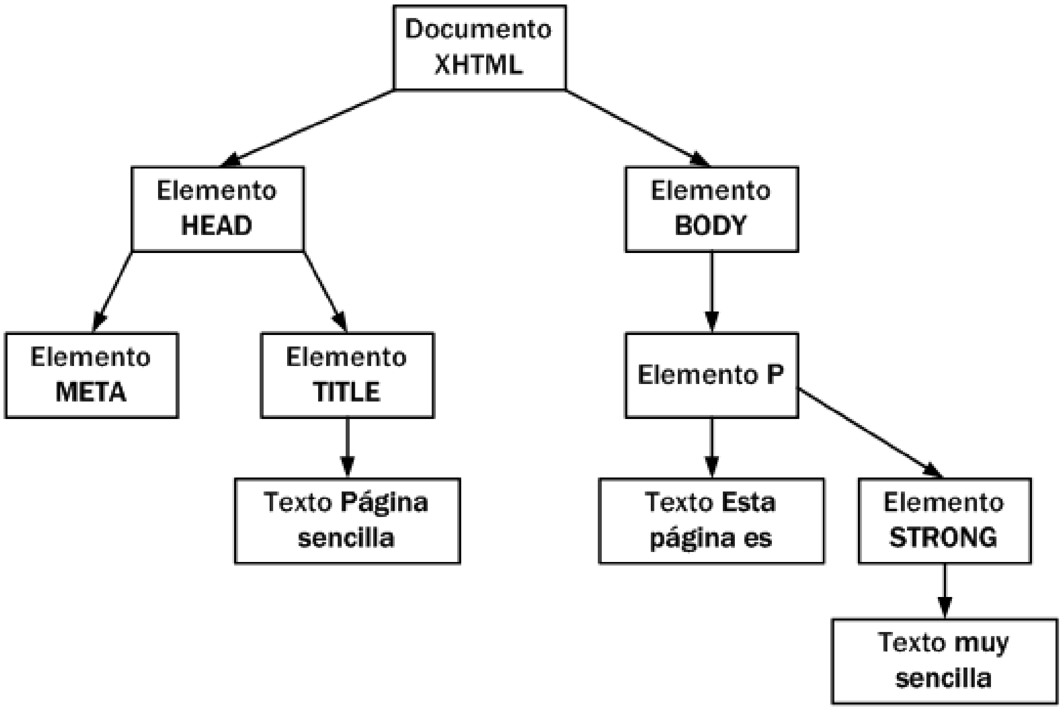


Figura 5.1. Árbol de nodos generado automáticamente por DOM a partir del código XHTML de la página

En el esquema anterior, cada rectángulo representa un nodo DOM y las flechas indican las relaciones entre nodos. Dentro de cada nodo, se ha incluido su tipo (que se verá más adelante) y su contenido.

La raíz del árbol de nodos de cualquier página XHTML siempre es la misma: un nodo de tipo especial denominado *"Documento"*.

A partir de ese nodo raíz, cada etiqueta XHTML se transforma en un nodo de tipo *"Elemento"*. La conversión de etiquetas en nodos se realiza de forma jerárquica. De esta forma, del nodo raíz solamente pueden derivar los nodos `HEAD` y `BODY`. A partir de esta derivación inicial, cada etiqueta XHTML se transforma en un nodo que deriva del nodo correspondiente a su *"etiqueta padre"*.

La transformación de las etiquetas XHTML habituales genera dos nodos: el primero es el nodo de tipo *"Elemento"* (correspondiente a la propia etiqueta XHTML) y el segundo es un nodo de tipo *"Texto"* que contiene el texto encerrado por esa etiqueta XHTML.

Así, la siguiente etiqueta XHTML:

```
<title>Página sencilla</title>
```

Genera los siguientes dos nodos:



Figura 5.2. Nodos generados automáticamente por DOM para una etiqueta XHTML sencilla

De la misma forma, la siguiente etiqueta XHTML:

```
<p>Esta página es <strong>muy sencilla</strong></p>
```

Genera los siguientes nodos:

- Nodo de tipo *"Elemento"* correspondiente a la etiqueta `<p>`.
- Nodo de tipo *"Texto"* con el contenido textual de la etiqueta `<p>`.
- Como el contenido de `<p>` incluye en su interior otra etiqueta XHTML, la etiqueta interior se transforma en un nodo de tipo *"Elemento"* que representa la etiqueta `<strong>` y que deriva del nodo anterior.
- El contenido de la etiqueta `<strong>` genera a su vez otro nodo de tipo *"Texto"* que deriva del nodo generado por `<strong>`.

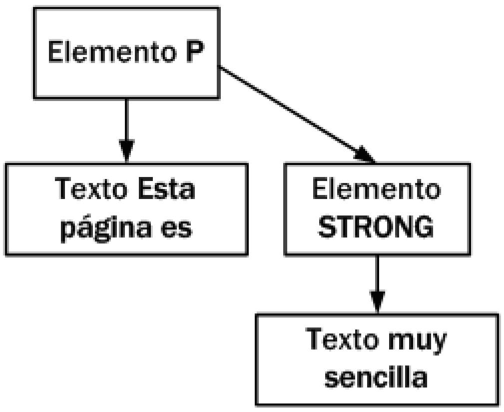


Figura 5.3. Nodos generados automáticamente por DOM para una etiqueta XHTML con otras etiquetas XHTML en su interior

La transformación automática de la página en un árbol de nodos siempre sigue las mismas reglas:

- Las etiquetas XHTML se transforman en dos nodos: el primero es la propia etiqueta y el segundo nodo es hijo del primero y consiste en el contenido textual de la etiqueta.
- Si una etiqueta XHTML se encuentra dentro de otra, se sigue el mismo procedimiento anterior, pero los nodos generados serán nodos hijo de su etiqueta padre.

Como se puede suponer, las páginas XHTML habituales producen árboles con miles de nodos. Aun así, el proceso de transformación es rápido y automático, siendo las funciones proporcionadas por DOM (que se verán más adelante) las únicas que permiten acceder a cualquier nodo de la página de forma sencilla e inmediata.

5.2. Tipos de nodos

La especificación completa de DOM define 12 tipos de nodos, aunque las páginas XHTML habituales se pueden manipular manejando solamente cuatro o cinco tipos de nodos:

- `Document`, nodo raíz del que derivan todos los demás nodos del árbol.
- `Element`, representa cada una de las etiquetas XHTML. Se trata del único nodo que puede contener atributos y el único del que pueden derivar otros nodos.
- `Attr`, se define un nodo de este tipo para representar cada uno de los atributos de las etiquetas XHTML, es decir, uno por cada par `atributo=valor`.
- `Text`, nodo que contiene el texto encerrado por una etiqueta XHTML.
- `Comment`, representa los comentarios incluidos en la página XHTML.

Los otros tipos de nodos existentes que no se van a considerar son `DocumentType`, `CDataSection`, `DocumentFragment`, `Entity`, `EntityReference`, `ProcessingInstruction` y `Notation`.

5.3. Acceso directo a los nodos

Una vez construido automáticamente el árbol completo de nodos DOM, ya es posible utilizar las funciones DOM para acceder de forma directa a cualquier nodo del árbol. Como acceder a un nodo del árbol es equivalente a acceder a *"un trozo"* de la página, una vez construido el árbol, ya es posible manipular de forma sencilla la página: acceder al valor de un elemento, establecer el valor de un elemento, mover un elemento de la página, crear y añadir nuevos elementos, etc.

DOM proporciona dos métodos alternativos para acceder a un nodo específico: acceso a través de sus nodos padre y acceso directo.

Las funciones que proporciona DOM para acceder a un nodo a través de sus nodos padre consisten en acceder al nodo raíz de la página y después a sus nodos hijos y a los nodos hijos de esos hijos y así sucesivamente hasta el último nodo de la rama terminada por el nodo buscado. Sin embargo, cuando se quiere acceder a un nodo específico, es mucho más rápido acceder directamente a ese nodo y no llegar hasta él descendiendo a través de todos sus nodos padre.

Por ese motivo, no se van a presentar las funciones necesarias para el acceso jerárquico de nodos y se muestran solamente las que permiten acceder de forma directa a los nodos.

Por último, es importante recordar que el acceso a los nodos, su modificación y su eliminación solamente es posible cuando el árbol DOM ha sido construido completamente, es decir, después de que la página XHTML se cargue por completo. Más adelante se verá cómo asegurar que un código JavaScript solamente se ejecute cuando el navegador ha cargado entera la página XHTML.

5.3.1. getElementsByTagName()

Como sucede con todas las funciones que proporciona DOM, la función `getElementsByTagName()` tiene un nombre muy largo, pero que lo hace autoexplicativo.

La función `getElementsByTagName(nombreEtiqueta)` obtiene todos los elementos de la página XHTML cuya etiqueta sea igual que el parámetro que se le pasa a la función.

El siguiente ejemplo muestra cómo obtener todos los párrafos de una página XHTML:

```
var parrafos = document.getElementsByTagName("p");
```

El valor que se indica delante del nombre de la función (en este caso, `document`) es el nodo a partir del cual se realiza la búsqueda de los elementos. En este caso, como se quieren obtener todos los párrafos de la página, se utiliza el valor `document` como punto de partida de la búsqueda.

El valor que devuelve la función es un array con todos los nodos que cumplen la condición de que su etiqueta coincide con el parámetro proporcionado. El valor devuelto es un array de nodos DOM, no un array de cadenas de texto o un array de objetos normales. Por lo tanto, se debe procesar cada valor del array de la forma que se muestra en las siguientes secciones.

De este modo, se puede obtener el primer párrafo de la página de la siguiente manera:

```
var primerParrafo = parrafos[0];
```

De la misma forma, se podrían recorrer todos los párrafos de la página con el siguiente código:

```
for(var i=0; i<parrafos.length; i++) {
  var parrafo = parrafos[i];
}
```

La función `getElementsByTagName()` se puede aplicar de forma recursiva sobre cada uno de los nodos devueltos por la función. En el siguiente ejemplo, se obtienen todos los enlaces del primer párrafo de la página:

```
var parrafos = document.getElementsByTagName("p");
var primerParrafo = parrafos[0];
var enlaces = primerParrafo.getElementsByTagName("a");
```

5.3.2. getElementsByName()

La función `getElementsByName()` es similar a la anterior, pero en este caso se buscan los elementos cuyo atributo `name` sea igual al parámetro proporcionado. En el siguiente ejemplo, se obtiene directamente el único párrafo con el nombre indicado:

```
var parrafoEspecial = document.getElementsByName("especial");

<p name="prueba">...</p>
<p name="especial">...</p>
<p>...</p>
```

Normalmente el atributo `name` es único para los elementos HTML que lo definen, por lo que es un método muy práctico para acceder directamente al nodo deseado. En el caso de los elementos HTML *radiobutton*, el atributo `name` es común a todos los *radiobutton* que están relacionados, por lo que la función devuelve una colección de elementos.

Internet Explorer 6.0 no implementa de forma correcta esta función, ya que sólo la tiene en cuenta para los elementos de tipo `<input>` y `<img>`. Además, también tiene en consideración los elementos cuyo atributo `id` sea igual al parámetro de la función.

5.3.3. getElementById()

La función `getElementById()` es la más utilizada cuando se desarrollan aplicaciones web dinámicas. Se trata de la función preferida para acceder directamente a un nodo y poder leer o modificar sus propiedades.

La función `getElementById()` devuelve el elemento XHTML cuyo atributo `id` coincide con el parámetro indicado en la función. Como el atributo `id` debe ser único para cada elemento de una misma página, la función devuelve únicamente el nodo deseado.

```
var cabecera = document.getElementById("cabecera");

<div id="cabecera">
  <a href="/" id="logo">...</a>
</div>
```

La función `getElementById()` es tan importante y tan utilizada en todas las aplicaciones web, que casi todos los ejemplos y ejercicios que siguen la utilizan constantemente.

Internet Explorer 6.0 también interpreta incorrectamente esta función, ya que devuelve también aquellos elementos cuyo atributo `name` coincida con el parámetro proporcionado a la función.

5.4. Creación y eliminación de nodos

Acceder a los nodos y a sus propiedades (que se verá más adelante) es sólo una parte de las manipulaciones habituales en las páginas. Las otras operaciones habituales son las de crear y eliminar nodos del árbol DOM, es decir, crear y eliminar *"trozos"* de la página web.

5.4.1. Creación de elementos XHTML simples

Como se ha visto, un elemento XHTML sencillo, como por ejemplo un párrafo, genera dos nodos: el primer nodo es de tipo `Element` y representa la etiqueta `<p>` y el segundo nodo es de tipo `Text` y representa el contenido textual de la etiqueta `<p>`.

Por este motivo, crear y añadir a la página un nuevo elemento XHTML sencillo consta de cuatro pasos diferentes:

1. Creación de un nodo de tipo `Element` que represente al elemento.
2. Creación de un nodo de tipo `Text` que represente el contenido del elemento.
3. Añadir el nodo `Text` como nodo hijo del nodo `Element`.
4. Añadir el nodo `Element` a la página, en forma de nodo hijo del nodo correspondiente al lugar en el que se quiere insertar el elemento.

De este modo, si se quiere añadir un párrafo simple al final de una página XHTML, es necesario incluir el siguiente código JavaScript:

```
// Crear nodo de tipo Element
var parrafo = document.createElement("p");

// Crear nodo de tipo Text
var contenido = document.createTextNode("Hola Mundo!");

// Añadir el nodo Text como hijo del nodo Element
parrafo.appendChild(contenido);

// Añadir el nodo Element como hijo de la pagina
document.body.appendChild(parrafo);
```

El proceso de creación de nuevos nodos puede llegar a ser tedioso, ya que implica la utilización de tres funciones DOM:

- `createElement(etiqueta)`: crea un nodo de tipo `Element` que representa al elemento XHTML cuya etiqueta se pasa como parámetro.
- `createTextNode(contenido)`: crea un nodo de tipo `Text` que almacena el contenido textual de los elementos XHTML.
- `nodoPadre.appendChild(nodoHijo)`: añade un nodo como hijo de otro nodo. Se debe utilizar al menos dos veces con los nodos habituales: en primer lugar se añade el nodo `Text` como hijo del nodo `Element` y a continuación se añade el nodo `Element` como hijo de algún nodo de la página.

5.4.2. Eliminación de nodos

Afortunadamente, eliminar un nodo del árbol DOM de la página es mucho más sencillo que añadirlo. En este caso, solamente es necesario utilizar la función `removeChild()`:

```
var parrafo = document.getElementById("provisional");
parrafo.parentNode.removeChild(parrafo);
```

```
<p id="provisional">...</p>
```

La función removeChild() requiere como parámetro el nodo que se va a eliminar. Además, esta función debe ser invocada desde el elemento padre de ese nodo que se quiere eliminar. La forma más segura y rápida de acceder al nodo padre de un elemento es mediante la propiedad nodoHijo.parentNode.

Así, para eliminar un nodo de una página XHTML se invoca a la función removeChild() desde el valor parentNode del nodo que se quiere eliminar. Cuando se elimina un nodo, también se eliminan automáticamente todos los nodos hijos que tenga, por lo que no es necesario borrar manualmente cada nodo hijo.

5.5. Acceso directo a los atributos XHTML

Una vez que se ha accedido a un nodo, el siguiente paso natural consiste en acceder y/o modificar sus atributos y propiedades. Mediante DOM, es posible acceder de forma sencilla a todos los atributos XHTML y todas las propiedades CSS de cualquier elemento de la página.

Los atributos XHTML de los elementos de la página se transforman automáticamente en propiedades de los nodos. Para acceder a su valor, simplemente se indica el nombre del atributo XHTML detrás del nombre del nodo.

El siguiente ejemplo obtiene de forma directa la dirección a la que enlaza el enlace:

```
var enlace = document.getElementById("enlace");
alert(enlace.href); // muestra http://www...com

<a id="enlace" href="http://www...com">Enlace</a>
```

En el ejemplo anterior, se obtiene el nodo DOM que representa el enlace mediante la función document.getElementById(). A continuación, se obtiene el atributo href del enlace mediante enlace.href. Para obtener por ejemplo el atributo id, se utilizaría enlace.id.

Las propiedades CSS no son tan fáciles de obtener como los atributos XHTML. Para obtener el valor de cualquier propiedad CSS del nodo, se debe utilizar el atributo style. El siguiente ejemplo obtiene el valor de la propiedad margin de la imagen:

```
var imagen = document.getElementById("imagen");
alert(imagen.style.margin);

<img id="imagen" style="margin:0; border:0;" src="logo.png" />
```

Aunque el funcionamiento es homogéneo entre distintos navegadores, los resultados no son exactamente iguales, como muestran las siguientes imágenes que son el resultado de ejecutar el código anterior en distintos navegadores:

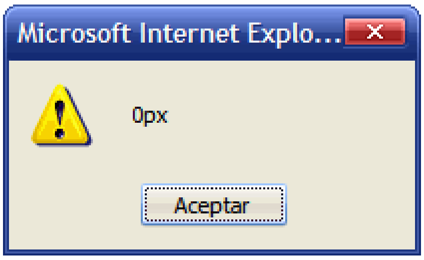


Figura 5.4. Valores que muestra Internet Explorer al acceder a las propiedades CSS a través de JavaScript

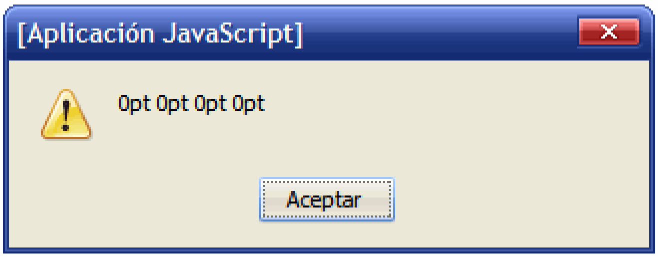


Figura 5.5. Valores que muestra Firefox al acceder a las propiedades CSS a través de JavaScript

Si el nombre de una propiedad CSS es compuesto, se accede a su valor modificando ligeramente su nombre:

```
var parrafo = document.getElementById("parrafo");
alert(parrafo.style.fontWeight);  // muestra "bold"

<p id="parrafo" style="font-weight: bold;">...</p>
```

La transformación del nombre de las propiedades CSS compuestas consiste en eliminar todos los guiones medios (-) y escribir en mayúscula la letra siguiente a cada guión medio. A continuación se muestran algunos ejemplos:

- `font-weight` se transforma en `fontWeight`
- `line-height` se transforma en `lineHeight`
- `border-top-style` se transforma en `borderTopStyle`
- `list-style-image` se transforma en `listStyleImage`

El único atributo XHTML que no tiene el mismo nombre en XHTML y en las propiedades DOM es el atributo `class`. Como la palabra `class` está reservada por JavaScript, no es posible utilizarla para acceder al atributo `class` del elemento XHTML. En su lugar, DOM utiliza el nombre `className` para acceder al atributo `class` de XHTML:

```
var parrafo = document.getElementById("parrafo");
alert(parrafo.class); // muestra "undefined"
alert(parrafo.className); // muestra "normal"

<p id="parrafo" class="normal">...</p>
```

5.6. Ejercicios sobre DOM

Ejercicio 11

A partir de la página web proporcionada y utilizando las funciones DOM, mostrar por pantalla la siguiente información:

1. Número de enlaces de la página
2. Dirección a la que enlaza el penúltimo enlace
3. Numero de enlaces que enlazan a http://prueba
4. Número de enlaces del tercer párrafo

Ejercicio 12

Completar el código JavaScript proporcionado para que cuando se pinche sobre el enlace se muestre completo el contenido de texto. Además, el enlace debe dejar de mostrarse después de pulsarlo por primera vez. La acción de pinchar sobre un enlace forma parte de los "Eventos" de JavaScript que se ven en el siguiente capítulo. En este ejercicio, sólo se debe saber que al pinchar sobre el enlace, se ejecuta la función llamada `muestra()`.

Ejercicio 13

Completar el código JavaScript proporcionado para que se añadan nuevos elementos a la lista cada vez que se pulsa sobre el botón. Utilizar las funciones DOM para crear nuevos nodos y añadirlos a la lista existente. Al igual que sucede en el ejercicio anterior, la acción de pinchar sobre un botón forma parte de los "Eventos" de JavaScript que se ven en el siguiente capítulo. En este ejercicio, sólo se debe saber que al pinchar sobre el botón, se ejecuta la función llamada `anade()`.

Capítulo 6. Eventos

Hasta ahora, todas las aplicaciones y scripts que se han creado tienen algo en común: se ejecutan desde la primera instrucción hasta la última de forma secuencial. Gracias a las estructuras de control de flujo (`if`, `for`, `while`) es posible modificar ligeramente este comportamiento y repetir algunos trozos del script y saltarse otros trozos en función de algunas condiciones.

Este tipo de aplicaciones son poco útiles, ya que no interactúan con los usuarios y no pueden responder a los diferentes *eventos* que se producen durante la ejecución de una aplicación. Afortunadamente, las aplicaciones web creadas con el lenguaje JavaScript pueden utilizar el modelo de *programación basada en eventos*.

En este tipo de programación, los scripts se dedican a esperar a que el usuario *"haga algo"* (que pulse una tecla, que mueva el ratón, que cierre la ventana del navegador). A continuación, el script responde a la acción del usuario normalmente procesando esa información y generando un resultado.

Los eventos hacen posible que los usuarios transmitan información a los programas. JavaScript define numerosos eventos que permiten una interacción completa entre el usuario y las páginas/aplicaciones web. La pulsación de una tecla constituye un evento, así como pinchar o mover el ratón, seleccionar un elemento de un formulario, redimensionar la ventana del navegador, etc.

JavaScript permite asignar una función a cada uno de los eventos. De esta forma, cuando se produce cualquier evento, JavaScript ejecuta su función asociada. Este tipo de funciones se denominan *"event handlers"* en inglés y suelen traducirse por *"manejadores de eventos"*.

6.1. Modelos de eventos

Crear páginas y aplicaciones web siempre ha sido mucho más complejo de lo que debería serlo debido a las incompatibilidades entre navegadores. A pesar de que existen decenas de estándares para las tecnologías empleadas, los navegadores no los soportan completamente o incluso los ignoran.

Las principales incompatibilidades se producen en el lenguaje XHTML, en el soporte de hojas de estilos CSS y sobre todo, en la implementación de JavaScript. De todas ellas, la incompatibilidad más importante se da precisamente en el modelo de eventos del navegador. Así, existen hasta tres modelos diferentes para manejar los eventos dependiendo del navegador en el que se ejecute la aplicación.

6.1.1. Modelo básico de eventos

Este modelo simple de eventos se introdujo para la versión 4 del estándar HTML y se considera parte del nivel más básico de DOM. Aunque sus características son limitadas,

es el único modelo que es compatible en todos los navegadores y por tanto, el único que permite crear aplicaciones que funcionan de la misma manera en todos los navegadores.

6.1.2. Modelo de eventos estándar

Las versiones más avanzadas del estándar DOM (DOM nivel 2) definen un modelo de eventos completamente nuevo y mucho más poderoso que el original. Todos los navegadores modernos lo incluyen, salvo Internet Explorer.

6.1.3. Modelo de eventos de Internet Explorer

Internet Explorer utiliza su propio modelo de eventos, que es similar pero incompatible con el modelo estándar. Se utilizó por primera vez en Internet Explorer 4 y Microsoft decidió seguir utilizándolo en el resto de versiones, a pesar de que la empresa había participado en la creación del estándar de DOM que define el modelo de eventos estándar.

6.2. Modelo básico de eventos

6.2.1. Tipos de eventos

En este modelo, cada elemento o etiqueta XHTML define su propia lista de posibles eventos que se le pueden asignar. Un mismo tipo de evento (por ejemplo, pinchar el botón izquierdo del ratón) puede estar definido para varios elementos XHTML diferentes y un mismo elemento XHTML puede tener asociados varios eventos diferentes.

El nombre de cada evento se construye mediante el prefijo on, seguido del nombre en inglés de la acción asociada al evento. Así, el evento de pinchar un elemento con el ratón se denomina `onclick` y el evento asociado a la acción de mover el ratón se denomina `onmousemove`.

La siguiente tabla resume los eventos más importantes definidos por JavaScript:

Evento	Descripción	Elementos para los que está definido
`onblur`	Deseleccionar el elemento	`<button>`, `<input>`, `<label>`, `<select>`, `<textarea>`, `<body>`
`onchange`	Deseleccionar un elemento que se ha modificado	`<input>`, `<select>`, `<textarea>`
`onclick`	Pinchar y soltar el ratón	Todos los elementos
`ondblclick`	Pinchar dos veces seguidas con el ratón	Todos los elementos
`onfocus`	Seleccionar un elemento	`<button>`, `<input>`, `<label>`, `<select>`, `<textarea>`, `<body>`
`onkeydown`	Pulsar una tecla (sin soltar)	Elementos de formulario y `<body>`
`onkeypress`	Pulsar una tecla	Elementos de formulario y `<body>`

onkeyup	Soltar una tecla pulsada	Elementos de formulario y <body>
onload	La página se ha cargado completamente	<body>
onmousedown	Pulsar (sin soltar) un botón del ratón	Todos los elementos
onmousemove	Mover el ratón	Todos los elementos
onmouseout	El ratón *"sale"* del elemento (pasa por encima de otro elemento)	Todos los elementos
onmouseover	El ratón *"entra"* en el elemento (pasa por encima del elemento)	Todos los elementos
onmouseup	Soltar el botón que estaba pulsado en el ratón	Todos los elementos
onreset	Inicializar el formulario (borrar todos sus datos)	<form>
onresize	Se ha modificado el tamaño de la ventana del navegador	<body>
onselect	Seleccionar un texto	<input>, <textarea>
onsubmit	Enviar el formulario	<form>
onunload	Se abandona la página (por ejemplo al cerrar el navegador)	<body>

Los eventos más utilizados en las aplicaciones web tradicionales son onload para esperar a que se cargue la página por completo, los eventos onclick, onmouseover, onmouseout para controlar el ratón y onsubmit para controlar el envío de los formularios.

Algunos eventos de la tabla anterior (onclick, onkeydown, onkeypress, onreset, onsubmit) permiten evitar la *"acción por defecto"* de ese evento. Más adelante se muestra en detalle este comportamiento, que puede resultar muy útil en algunas técnicas de programación.

Las acciones típicas que realiza un usuario en una página web pueden dar lugar a una sucesión de eventos. Al pulsar por ejemplo sobre un botón de tipo <input type="submit"> se desencadenan los eventos onmousedown, onclick, onmouseup y onsubmit de forma consecutiva.

6.2.2. Manejadores de eventos

Un evento de JavaScript por sí mismo carece de utilidad. Para que los eventos resulten útiles, se deben asociar funciones o código JavaScript a cada evento. De esta forma, cuando se produce un evento se ejecuta el código indicado, por lo que la aplicación puede *responder* ante cualquier evento que se produzca durante su ejecución.

Las funciones o código JavaScript que se definen para cada evento se denominan *"manejador de eventos"* y como JavaScript es un lenguaje muy flexible, existen varias formas diferentes de indicar los manejadores:

- Manejadores como atributos de los elementos XHTML.
- Manejadores como funciones JavaScript externas.
- Manejadores *"semánticos"*.

6.2.2.1. Manejadores de eventos como atributos XHTML

Se trata del método más sencillo y a la vez *menos profesional* de indicar el código JavaScript que se debe ejecutar cuando se produzca un evento. En este caso, el código se incluye en un atributo del propio elemento XHTML. En el siguiente ejemplo, se quiere mostrar un mensaje cuando el usuario pinche con el ratón sobre un botón:

```
<input type="button" value="Pinchame y verás" onclick="alert('Gracias por
pinchar');" />
```

En este método, se definen atributos XHTML con el mismo nombre que los eventos que se quieren manejar. El ejemplo anterior sólo quiere controlar el evento de pinchar con el ratón, cuyo nombre es `onclick`. Así, el elemento XHTML para el que se quiere definir este evento, debe incluir un atributo llamado `onclick`.

El contenido del atributo es una cadena de texto que contiene todas las instrucciones JavaScript que se ejecutan cuando se produce el evento. En este caso, el código JavaScript es muy sencillo (`alert('Gracias por pinchar');`), ya que solamente se trata de mostrar un mensaje.

En este otro ejemplo, cuando el usuario pincha sobre el elemento `<div>` se muestra un mensaje y cuando el usuario pasa el ratón por encima del elemento, se muestra otro mensaje:

```
<div onclick="alert('Has pinchado con el ratón');" onmouseover="alert('Acabas
de pasar el ratón por encima');">
  Puedes pinchar sobre este elemento o simplemente pasar el ratón por encima
</div>
```

Este otro ejemplo incluye una de las instrucciones más utilizadas en las aplicaciones JavaScript más antiguas:

```
<body onload="alert('La página se ha cargado completamente');">
   ...
</body>
```

El mensaje anterior se muestra después de que la página se haya cargado completamente, es decir, después de que se haya descargado su código HTML, sus imágenes y cualquier otro objeto incluido en la página.

El evento `onload` es uno de los más utilizados ya que, como se vio en el capítulo de DOM, las funciones que permiten acceder y manipular los nodos del árbol DOM solamente están disponibles cuando la página se ha cargado completamente.

6.2.2.2. Manejadores de eventos y variable this

JavaScript define una variable especial llamada `this` que se crea automáticamente y que se emplea en algunas técnicas avanzadas de programación. En los eventos, se puede utilizar la variable `this` para referirse al elemento XHTML que ha provocado el evento. Esta variable es muy útil para ejemplos como el siguiente:

Cuando el usuario pasa el ratón por encima del `<div>`, el color del borde se muestra de color negro. Cuando el ratón *sale* del `<div>`, se vuelve a mostrar el borde con el color gris claro original.

Elemento `<div>` original:

```
<div id="contenidos" style="width:150px; height:60px; border:thin solid silver">
  Sección de contenidos...
</div>
```

Si no se utiliza la variable `this`, el código necesario para modificar el color de los bordes, sería el siguiente:

```
<div id="contenidos" style="width:150px; height:60px; border:thin solid silver"
onmouseover="document.getElementById('contenidos').style.borderColor='black';"
onmouseout="document.getElementById('contenidos').style.borderColor='silver';">
  Sección de contenidos...
</div>
```

El código anterior es demasiado largo y demasiado propenso a cometer errores. Dentro del código de un evento, JavaScript crea automáticamente la variable `this`, que hace referencia al elemento XHTML que ha provocado el evento. Así, el ejemplo anterior se puede reescribir de la siguiente manera:

```
<div id="contenidos" style="width:150px; height:60px; border:thin solid silver"
onmouseover="this.style.borderColor='black';"
onmouseout="this.style.borderColor='silver';">
  Sección de contenidos...
</div>
```

El código anterior es mucho más compacto, más fácil de leer y de escribir y sigue funcionando correctamente aunque se modifique el valor del atributo `id` del `<div>`.

6.2.2.3. Manejadores de eventos como funciones externas

La definición de los manejadores de eventos en los atributos XHTML es el método más sencillo pero menos aconsejable de tratar con los eventos en JavaScript. El principal inconveniente es que se complica en exceso en cuanto se añaden algunas pocas instrucciones, por lo que solamente es recomendable para los casos más sencillos.

Si se realizan aplicaciones complejas, como por ejemplo la validación de un formulario, es aconsejable agrupar todo el código JavaScript en una función externa y llamar a esta función desde el elemento XHTML.

Siguiendo con el ejemplo anterior que muestra un mensaje al pinchar sobre un botón:

```
<input type="button" value="Pinchame y verás" onclick="alert('Gracias por
pinchar');" />
```

Utilizando funciones externas se puede transformar en:

```
function muestraMensaje() {
  alert('Gracias por pinchar');
}

<input type="button" value="Pinchame y verás" onclick="muestraMensaje()" />
```

Esta técnica consiste en extraer todas las instrucciones de JavaScript y agruparlas en una función externa. Una vez definida la función, en el atributo del elemento XHTML se incluye el nombre de la función, para indicar que es la función que se ejecuta cuando se produce el evento.

La llamada a la función se realiza de la forma habitual, indicando su nombre seguido de los paréntesis y de forma opcional, incluyendo todos los argumentos y parámetros que se necesiten.

El principal inconveniente de este método es que en las funciones externas no se puede seguir utilizando la variable `this` y por tanto, es necesario pasar esta variable como parámetro a la función:

```
function resalta(elemento) {
  switch(elemento.style.borderColor) {
    case 'silver':
    case 'silver silver silver silver':
    case '#c0c0c0':
      elemento.style.borderColor = 'black';
      break;
    case 'black':
    case 'black black black black':
    case '#000000':
      elemento.style.borderColor = 'silver';
      break;
  }
}

<div style="width:150px; height:60px; border:thin solid silver"
onmouseover="resalta(this)" onmouseout="resalta(this)">
  Sección de contenidos...
</div>
```

En el ejemplo anterior, la función externa es llamada con el parámetro `this`, que dentro de la función se denomina `elemento`. La complejidad del ejemplo se produce sobre todo por la forma en la que los distintos navegadores almacenan el valor de la propiedad `borderColor`.

Mientras que Firefox almacena (en caso de que los cuatro bordes coincidan en color) el valor `black`, Internet Explorer lo almacena como `black black black black` y Opera almacena su representación hexadecimal `#000000`.

6.2.2.4. Manejadores de eventos semánticos

Los métodos que se han visto para añadir manejadores de eventos (como atributos XHTML y como funciones externas) tienen un grave inconveniente: *"ensucian"* el código XHTML de la página.

Como es conocido, una de las buenas prácticas básicas en el diseño de páginas y aplicaciones web es la separación de los contenidos (XHTML) y su aspecto o presentación (CSS). Siempre que sea posible, también se recomienda separar los contenidos (XHTML) y su comportamiento o programación (JavaScript).

Mezclar el código JavaScript con los elementos XHTML solamente contribuye a complicar el código fuente de la página, a dificultar la modificación y mantenimiento de la página y a reducir la semántica del documento final producido.

Afortunadamente, existe un método alternativo para definir los manejadores de eventos de JavaScript. Esta técnica es una evolución del método de las funciones externas, ya que se basa en utilizar las propiedades DOM de los elementos XHTML para asignar todas las funciones externas que actúan de manejadores de eventos. Así, el siguiente ejemplo:

```
<input id="pinchable" type="button" value="Pinchame y verás"
onclick="alert('Gracias por pinchar');" />
```

Se puede transformar en:

```
// Función externa
function muestraMensaje() {
  alert('Gracias por pinchar');
}

// Asignar la función externa al elemento
document.getElementById("pinchable").onclick = muestraMensaje;

// Elemento XHTML
<input id="pinchable" type="button" value="Pinchame y verás" />
```

La técnica de los manejadores semánticos consiste en:

1. Asignar un identificador único al elemento XHTML mediante el atributo `id`.
2. Crear una función de JavaScript encargada de manejar el evento.
3. Asignar la función externa al evento correspondiente en el elemento deseado.

El último paso es la clave de esta técnica. En primer lugar, se obtiene el elemento al que se desea asociar la función externa:

```
document.getElementById("pinchable");
```

A continuación, se utiliza una propiedad del elemento con el mismo nombre que el evento que se quiere manejar. En este caso, la propiedad es `onclick`:

```
document.getElementById("pinchable").onclick = ...
```

Por último, se asigna la función externa mediante su nombre sin paréntesis. Lo más importante (y la causa más común de errores) es indicar solamente el nombre de la función, es decir, prescindir de los paréntesis al asignar la función:

```
document.getElementById("pinchable").onclick = muestraMensaje;
```

Si se añaden los paréntesis después del nombre de la función, en realidad se está ejecutando la función y guardando el valor devuelto por la función en la propiedad `onclick` de elemento.

```
// Asignar una función externa a un evento de un elemento
document.getElementById("pinchable").onclick = muestraMensaje;

// Ejecutar una función y guardar su resultado en una propiedad de un elemento
document.getElementById("pinchable").onclick = muestraMensaje();
```

La gran ventaja de este método es que el código XHTML resultante es muy *"limpio"*, ya que no se mezcla con el código JavaScript. Además, dentro de las funciones externas asignadas sí que se puede utilizar la variable `this` para referirse al elemento que provoca el evento.

El único inconveniente de este método es que la página se debe cargar completamente antes de que se puedan utilizar las funciones DOM que asignan los manejadores a los elementos XHTML. Una de las formas más sencillas de asegurar que cierto código se va a ejecutar después de que la página se cargue por completo es utilizar el evento `onload`:

```
window.onload = function() {
  document.getElementById("pinchable").onclick = muestraMensaje;
}
```

La técnica anterior utiliza el concepto de *funciones anónimas*, que no se va a estudiar, pero que permite crear un código compacto y muy sencillo. Para asegurarse que un código JavaScript va a ejecutarse después de que la página se haya cargado completamente, sólo es necesario incluir esas instrucciones entre los símbolos { y }:

```
window.onload = function() {
  ...
}
```

En el siguiente ejemplo, se añaden eventos a los elementos de tipo `input=text` de un formulario complejo:

```
function resalta() {
  // Código JavaScript
}

window.onload = function() {
  var formulario = document.getElementById("formulario");
  var camposInput = formulario.getElementsByTagName("input");

  for(var i=0; i<camposInput.length; i++) {
    if(camposInput[i].type == "text") {
      camposInput[i].onclick = resalta;
    }
```

```
    }
}
```

Ejercicio 14

A partir de la página web proporcionada, completar el código JavaScript para que:

1. Cuando se pinche sobre el primer enlace, se oculte su sección relacionada
2. Cuando se vuelva a pinchar sobre el mismo enlace, se muestre otra vez esa sección de contenidos
3. Completar el resto de enlaces de la página para que su comportamiento sea idéntico al del primer enlace
4. Cuando una sección se oculte, debe cambiar el mensaje del enlace asociado (pista: propiedad `innerHTML`)

6.3. Obteniendo información del evento (objeto event)

Normalmente, los manejadores de eventos requieren información adicional para procesar sus tareas. Si una función por ejemplo se encarga de procesar el evento `onclick`, quizás necesite saber en que posición estaba el ratón en el momento de pinchar el botón.

No obstante, el caso más habitual en el que es necesario conocer información adicional sobre el evento es el de los eventos asociados al teclado. Normalmente, es muy importante conocer la tecla que se ha pulsado, por ejemplo para diferenciar las teclas normales de las teclas especiales (`ENTER`, tabulador, `Alt`, `Ctrl.`, etc.).

JavaScript permite obtener información sobre el ratón y el teclado mediante un objeto especial llamado `event`. Desafortunadamente, los diferentes navegadores presentan diferencias muy notables en el tratamiento de la información sobre los eventos.

La principal diferencia reside en la forma en la que se obtiene el objeto `event`. Internet Explorer considera que este objeto forma parte del objeto `window` y el resto de navegadores lo consideran como el único argumento que tienen las funciones manejadoras de eventos.

Aunque es un comportamiento que resulta muy extraño al principio, todos los navegadores modernos excepto Internet Explorer crean *mágicamente* y de forma automática un argumento que se pasa a la función manejadora, por lo que no es necesario incluirlo en la llamada a la función manejadora. De esta forma, para utilizar este *"argumento mágico"*, sólo es necesario asignarle un nombre, ya que los navegadores lo crean automáticamente.

En resumen, en los navegadores tipo Internet Explorer, el objeto `event` se obtiene directamente mediante:

```
var evento = window.event;
```

Por otra parte, en el resto de navegadores, el objeto `event` se obtiene *mágicamente* a partir del argumento que el navegador crea automáticamente:

```
function manejadorEventos(elEvento) {
  var evento = elEvento;
}
```

Si se quiere programar una aplicación que funcione correctamente en todos los navegadores, es necesario obtener el objeto event de forma correcta según cada navegador. El siguiente código muestra la forma correcta de obtener el objeto event en cualquier navegador:

```
function manejadorEventos(elEvento) {
  var evento = elEvento || window.event;
}
```

Una vez obtenido el objeto event, ya se puede acceder a toda la información relacionada con el evento, que depende del tipo de evento producido.

6.3.1. Información sobre el evento

La propiedad type indica el tipo de evento producido, lo que es útil cuando una misma función se utiliza para manejar varios eventos:

```
var tipo = evento.type;
```

La propiedad type devuelve el tipo de evento producido, que es igual al nombre del evento pero sin el prefijo on.

Mediante esta propiedad, se puede rehacer de forma más sencilla el ejemplo anterior en el que se resaltaba una sección de contenidos al pasar el ratón por encima:

```
function resalta(elEvento) {
  var evento = elEvento || window.event;
  switch(evento.type) {
    case 'mouseover':
      this.style.borderColor = 'black';
      break;
    case 'mouseout':
      this.style.borderColor = 'silver';
      break;
  }
}

window.onload = function() {
  document.getElementById("seccion").onmouseover = resalta;
  document.getElementById("seccion").onmouseout = resalta;
}

<div id="seccion" style="width:150px; height:60px; border:thin solid silver">
  Sección de contenidos...
</div>
```

6.3.2. Información sobre los eventos de teclado

De todos los eventos disponibles en JavaScript, los eventos relacionados con el teclado son los más incompatibles entre diferentes navegadores y por tanto, los más difíciles de

manejar. En primer lugar, existen muchas diferencias entre los navegadores, los teclados y los sistemas operativos de los usuarios, principalmente debido a las diferencias entre idiomas.

Además, existen tres eventos diferentes para las pulsaciones de las teclas (`onkeyup`, `onkeypress` y `onkeydown`). Por último, existen dos tipos de teclas: las teclas *normales* (como letras, números y símbolos normales) y las teclas *especiales* (como `ENTER`, `Alt`, `Shift`, etc.)

Cuando un usuario pulsa una tecla normal, se producen tres eventos seguidos y en este orden: `onkeydown`, `onkeypress` y `onkeyup`. El evento `onkeydown` se corresponde con el hecho de pulsar una tecla y no soltarla; el evento `onkeypress` es la propia pulsación de la tecla y el evento `onkeyup` hace referencia al hecho de soltar una tecla que estaba pulsada.

La forma más sencilla de obtener la información sobre la tecla que se ha pulsado es mediante el evento `onkeypress`. La información que proporcionan los eventos `onkeydown` y `onkeyup` se puede considerar como más técnica, ya que devuelven el código interno de cada tecla y no el carácter que se ha pulsado.

A continuación se incluye una lista con todas las propiedades diferentes de todos los eventos de teclado tanto en Internet Explorer como en el resto de navegadores:

- Evento `keydown`:
  - Mismo comportamiento en todos los navegadores:
    - Propiedad `keyCode`: código interno de la tecla
    - Propiedad `charCode`: no definido
- Evento `keypress`:
  - Internet Explorer:
    - Propiedad `keyCode`: el código del carácter de la tecla que se ha pulsado
    - Propiedad `charCode`: no definido
  - Resto de navegadores:
    - Propiedad `keyCode`: para las teclas normales, no definido. Para las teclas especiales, el código interno de la tecla.
    - Propiedad `charCode`: para las teclas normales, el código del carácter de la tecla que se ha pulsado. Para las teclas especiales, `0`.
- Evento `keyup`:
  - Mismo comportamiento en todos los navegadores:
    - Propiedad `keyCode`: código interno de la tecla
    - Propiedad `charCode`: no definido

Para convertir el código de un carácter (no confundir con el código interno) al carácter que representa la tecla que se ha pulsado, se utiliza la función `String.fromCharCode()`.

A continuación se incluye un script que muestra toda la información sobre los tres eventos de teclado:

```
window.onload = function() {
  document.onkeyup = muestraInformacion;
  document.onkeypress = muestraInformacion;
  document.onkeydown = muestraInformacion;
}

function muestraInformacion(elEvento) {
  var evento = window.event || elEvento;

  var mensaje = "Tipo de evento: " + evento.type + "<br>" +
                "Propiedad keyCode: " + evento.keyCode + "<br>" +
                "Propiedad charCode: " + evento.charCode + "<br>" +
                "Carácter pulsado: " + String.fromCharCode(evento.charCode);

  info.innerHTML += "<br>--------------------------------------<br>" + mensaje
}

...

<div id="info"></div>
```

Al pulsar la tecla a en el navegador Firefox, se muestra la siguiente sucesión de eventos:

```
--------------------------------------
Tipo de evento: keydown
Propiedad keyCode: 65
Propiedad charCode: 0
Carácter pulsado: ?
--------------------------------------
Tipo de evento: keypress
Propiedad keyCode: 0
Propiedad charCode: 97
Carácter pulsado: a
--------------------------------------
Tipo de evento: keyup
Propiedad keyCode: 65
Propiedad charCode: 0
Carácter pulsado: ?
```

Al pulsar la tecla `A` (la misma tecla, pero habiendo activado previamente las mayúsculas) se muestra la siguiente sucesión de eventos en el navegador Firefox:

```
--------------------------------------
Tipo de evento: keydown
Propiedad keyCode: 65
Propiedad charCode: 0
Carácter pulsado: ?
--------------------------------------
Tipo de evento: keypress
```

```
Propiedad keyCode: 0
Propiedad charCode: 65
Carácter pulsado: A
--------------------------------------
Tipo de evento: keyup
Propiedad keyCode: 65
Propiedad charCode: 0
Carácter pulsado: ?
```

En los eventos `keydown` y `keyup`, la propiedad `keyCode` sigue valiendo lo mismo en los dos casos. El motivo es que `keyCode` almacena el código interno de la tecla, por lo que si se pulsa la misma tecla, se obtiene el mismo código, independientemente de que una misma tecla puede producir caracteres diferentes (por ejemplo mayúsculas y minúsculas).

En el evento `keypress`, el valor de la propiedad `charCode` varía, ya que el carácter `a`, no es el mismo que el carácter `A`. En este caso, el valor de `charCode` coincide con el código ASCII del carácter pulsado.

Siguiendo en el navegador Firefox, si ahora se pulsa una tecla *especial*, como por ejemplo el tabulador, se muestra la siguiente información:

```
--------------------------------------
Tipo de evento: keydown
Propiedad keyCode: 9
Propiedad charCode: 0
Carácter pulsado: ?
--------------------------------------
Tipo de evento: keypress
Propiedad keyCode: 9
Propiedad charCode: 0
Carácter pulsado: ?
--------------------------------------
Tipo de evento: keyup
Propiedad keyCode: 9
Propiedad charCode: 0
Carácter pulsado: ?
```

Las teclas especiales no disponen de la propiedad `charCode`, ya que sólamente se guarda el código interno de la tecla pulsada en la propiedad `keyCode`, en este caso el código `9`. Si se pulsa la tecla `Enter`, se obtiene el código `13`, la tecla de la flecha superior produce el código `38`, etc. No obstante, dependiendo del teclado utilizado para pulsar las teclas y dependiendo de la disposición de las teclas en función del idioma del teclado, estos códigos podrían variar.

A continuación se muestra el resultado de la ejecución del mismo ejemplo anterior en el navegador Internet Explorer. Al pulsar la tecla `a`, se obtiene la siguiente información:

```
--------------------------------------
Tipo de evento: keydown
Propiedad keyCode: 65
Propiedad charCode: undefined
Carácter pulsado:
```

```
--------------------------------------
Tipo de evento: keypress
Propiedad keyCode: 97
Propiedad charCode: undefined
Carácter pulsado:
--------------------------------------
Tipo de evento: keyup
Propiedad keyCode: 65
Propiedad charCode: undefined
Carácter pulsado:
```

La propiedad `keyCode` en el evento `keypress` contiene el código ASCII del carácter de la tecla, por lo que se puede obtener directamente el carácter mediante `String.fromCharCode(keyCode)`.

Si se pulsa la tecla `A`, la información mostrada es idéntica a la anterior, salvo que el código que muestra el evento `keypress` cambia por `65`, que es el código ASCII de la tecla `A`:

```
--------------------------------------
Tipo de evento: keydown
Propiedad keyCode: 65
Propiedad charCode: undefined
Carácter pulsado:
--------------------------------------
Tipo de evento: keypress
Propiedad keyCode: 65
Propiedad charCode: undefined
Carácter pulsado:
--------------------------------------
Tipo de evento: keyup
Propiedad keyCode: 65
Propiedad charCode: undefined
Carácter pulsado:
```

Al pulsar una tecla *especial* como el tabulador, Internet Explorer muestra la siguiente información:

```
--------------------------------------
Tipo de evento: keydown
Propiedad keyCode: 9
Propiedad charCode: undefined
Carácter pulsado:
```

Los códigos mostrados para las teclas especiales coinciden con los de Firefox y el resto de navegadores, pero recuerda que pueden variar en función del teclado que se utiliza y en función de la disposición de las teclas para cada idioma.

Por último, las propiedades `altKey`, `ctrlKey` y `shiftKey` almacenan un valor booleano que indica si alguna de esas teclas estaba pulsada al producirse el evento del teclado. Sorprendentemente, estas tres propiedades funcionan de la misma forma en todos los navegadores:

```
if(evento.altKey) {
  alert('Estaba pulsada la tecla ALT');
}
```

A continuación se muestra el caso en el que se pulsa la tecla `Shift` y sin soltarla, se pulsa sobre la tecla que contiene el número `2` (en este caso, se refiere a la tecla que se encuentra en la parte superior del teclado y por tanto, no se refiere a la que se encuentra en el teclado numérico). Tanto Internet Explorer como Firefox muestran la misma secuencia de eventos:

```
--------------------------------------
Tipo de evento: keydown
Propiedad keyCode: 16
Propiedad charCode: 0
Carácter pulsado: ?
--------------------------------------
Tipo de evento: keydown
Propiedad keyCode: 50
Propiedad charCode: 0
Carácter pulsado: ?
--------------------------------------
Tipo de evento: keypress
Propiedad keyCode: 0
Propiedad charCode: 34
Carácter pulsado: "
--------------------------------------
Tipo de evento: keyup
Propiedad keyCode: 50
Propiedad charCode: 0
Carácter pulsado: ?
--------------------------------------
Tipo de evento: keyup
Propiedad keyCode: 16
Propiedad charCode: 0
Carácter pulsado: ?
```

El evento `keypress` es el único que permite obtener el *carácter realmente pulsado*, ya que al pulsar sobre la tecla `2` habiendo pulsado la tecla `Shift` previamente, se obtiene el carácter ", que es precisamente el que muestra el evento `keypress`.

El siguiente código de JavaScript permite obtener de forma correcta en cualquier navegador el carácter correspondiente a la tecla pulsada:

```
function manejador(elEvento) {
  var evento = elEvento || window.event;
  var caracter = evento.charCode || evento.keyCode;
  alert("El carácter pulsado es: " + String.fromCharCode(caracter));
}

document.onkeypress = manejador;
```

6.3.3. Información sobre los eventos de ratón

La información más relevante sobre los eventos relacionados con el ratón es la de las coordenadas de la posición del puntero del ratón. Aunque el origen de las coordenadas siempre se encuentra en la esquina superior izquierda, el punto que se toma como referencia de las coordenadas puede variar.

De esta forma, es posible obtener la posición del ratón respecto de la pantalla del ordenador, respecto de la ventana del navegador y respecto de la propia página HTML (que se utiliza cuando el usuario ha hecho *scroll* sobre la página). Las coordenadas más sencillas son las que se refieren a la posición del puntero respecto de la ventana del navegador, que se obtienen mediante las propiedades `clientX` y `clientY`:

```
function muestraInformacion(elEvento) {
  var evento = elEvento || window.event;
  var coordenadaX = evento.clientX;
  var coordenadaY = evento.clientY;
  alert("Has pulsado el ratón en la posición: " + coordenadaX + ", " +
coordenadaY);
}

document.onclick = muestraInformacion;
```

Las coordenadas de la posición del puntero del ratón respecto de la pantalla completa del ordenador del usuario se obtienen de la misma forma, mediante las propiedades `screenX` y `screenY`:

```
var coordenadaX = evento.screenX;
var coordenadaY = evento.screenY;
```

En muchas ocasiones, es necesario obtener otro par de coordenadas diferentes: las que corresponden a la posición del ratón respecto del origen de la página. Estas coordenadas no siempre coinciden con las coordenadas respecto del origen de la ventana del navegador, ya que el usuario puede hacer *scroll* sobre la página web. Internet Explorer no proporciona estas coordenadas de forma directa, mientras que el resto de navegadores sí que lo hacen. De esta forma, es necesario detectar si el navegador es de tipo Internet Explorer y en caso afirmativo realizar un cálculo sencillo:

```
// Detectar si el navegador es Internet Explorer
var ie = navigator.userAgent.toLowerCase().indexOf('msie')!=-1;

if(ie) {
  coordenadaX = evento.clientX + document.body.scrollLeft;
  coordenadaY = evento.clientY + document.body.scrollTop;
}
else {
  coordenadaX = evento.pageX;
  coordenadaY = evento.pageY;
}
alert("Has pulsado el ratón en la posición: " + coordenadaX + ", " +
coordenadaY + " respecto de la página web");
```

La variable `ie` vale `true` si el navegador en el que se ejecuta el script es de tipo Internet Explorer (cualquier versión) y vale `false` en otro caso. Para el resto de navegadores, las coordenadas respecto del origen de la página se obtienen mediante las propiedades `pageX` y `pageY`. En el caso de Internet Explorer, se obtienen sumando la posición respecto de la ventana del navegador (`clientX`, `clientY`) y el desplazamiento que ha sufrido la página (`document.body.scrollLeft`, `document.body.scrollTop`).

Ejercicio 15

Completar el código JavaScript proporcionado para que:

1. Al mover el ratón en cualquier punto de la ventana del navegador, se muestre la posición del puntero respecto del navegador y respecto de la página:

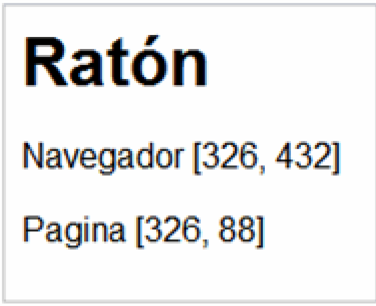


Figura 6.1. Información que se muestra para los eventos del ratón

Para mostrar los mensajes, utilizar la función `muestraInformacion()` deduciendo su funcionamiento a partir de su código fuente.

2. Al pulsar cualquier tecla, el mensaje mostrado debe cambiar para indicar el nuevo evento y su información asociada:



Figura 6.2. Información que se muestra para los eventos del teclado

3. Añadir la siguiente característica al script: cuando se pulsa un botón del ratón, el color de fondo del cuadro de mensaje debe ser amarillo (`#FFFFCC`) y cuando se pulsa una tecla, el color de fondo debe ser azul (`#CCE6FF`). Al volver a mover el ratón, el color de fondo vuelve a ser blanco.

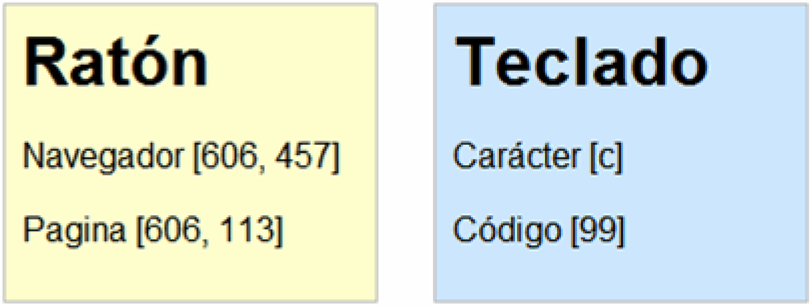


Figura 6.3. El color de fondo del cuadro de información cambia en función del tipo de evento

Ejercicio 16

Crear un script que informe al usuario en que zona de la pantalla ha pulsado el ratón. Las zonas definidas son las siguientes: izquierda arriba, izquierda abajo, derecha arriba y derecha abajo. Para determinar el tamaño de la ventana del navegador, utilizar la función `tamanoVentanaNavegador()` proporcionada.

Capítulo 7. Formularios

La programación de aplicaciones que contienen formularios web siempre ha sido una de las tareas fundamentales de JavaScript. De hecho, una de las principales razones por las que se inventó el lenguaje de programación JavaScript fue la necesidad de validar los datos de los formularios directamente en el navegador del usuario. De esta forma, se evitaba recargar la página cuando el usuario cometía errores al rellenar los formularios.

No obstante, la aparición de las aplicaciones AJAX ha relevado al tratamiento de formularios como la principal actividad de JavaScript. Ahora, el principal uso de JavaScript es el de las comunicaciones asíncronas con los servidores y el de la manipulación dinámica de las aplicaciones. De todas formas, el manejo de los formularios sigue siendo un requerimiento imprescindible para cualquier programador de JavaScript.

7.1. Propiedades básicas de formularios y elementos

JavaScript dispone de numerosas propiedades y funciones que facilitan la programación de aplicaciones que manejan formularios. En primer lugar, cuando se carga una página web, el navegador crea automáticamente un array llamado `forms` y que contiene la referencia a todos los formularios de la página.

Para acceder al array `forms`, se utiliza el objeto `document`, por lo que `document.forms` es el array que contiene todos los formularios de la página. Como se trata de un array, el acceso a cada formulario se realiza con la misma sintaxis de los arrays. La siguiente instrucción accede al primer formulario de la página:

```
document.forms[0];
```

Además del array de formularios, el navegador crea automáticamente un array llamado `elements` por cada uno de los formularios de la página. Cada array `elements` contiene la referencia a todos los elementos (cuadros de texto, botones, listas desplegables, etc.) de ese formulario. Utilizando la sintaxis de los arrays, la siguiente instrucción obtiene el primer elemento del primer formulario de la página:

```
document.forms[0].elements[0];
```

La sintaxis de los arrays no siempre es tan concisa. El siguiente ejemplo muestra cómo obtener directamente el último elemento del primer formulario de la página:

```
document.forms[0].elements[document.forms[0].elements.length-1];
```

Aunque esta forma de acceder a los formularios es rápida y sencilla, tiene un inconveniente muy grave. ¿Qué sucede si cambia el diseño de la página y en el código HTML se cambia el orden de los formularios originales o se añaden nuevos formularios? El problema es que *"el primer formulario de la página"* ahora podría ser otro formulario diferente al que espera la aplicación.

En un entorno tan cambiante como el diseño web, es muy difícil confiar en que el orden de los formularios se mantenga estable en una página web. Por este motivo, siempre debería evitarse el acceso a los formularios de una página mediante el array `document.forms`.

Una forma de evitar los problemas del método anterior consiste en acceder a los formularios de una página a través de su nombre (atributo `name`) o a través de su atributo `id`. El objeto `document` permite acceder directamente a cualquier formulario mediante su atributo `name`:

```
var formularioPrincipal = document.formulario;
var formularioSecundario = document.otro_formulario;

<form name="formulario" >
  ...
</form>

<form name="otro_formulario" >
  ...
</form>
```

Accediendo de esta forma a los formularios de la página, el script funciona correctamente aunque se reordenen los formularios o se añadan nuevos formularios a la página. Los elementos de los formularios también se pueden acceder directamente mediante su atributo `name`:

```
var formularioPrincipal = document.formulario;
var primerElemento = document.formulario.elemento;

<form name="formulario">
  <input type="text" name="elemento" />
</form>
```

Obviamente, también se puede acceder a los formularios y a sus elementos utilizando las funciones DOM de acceso directo a los nodos. El siguiente ejemplo utiliza la habitual función `document.getElementById()` para acceder de forma directa a un formulario y a uno de sus elementos:

```
var formularioPrincipal = document.getElementById("formulario");
var primerElemento = document.getElementById("elemento");

<form name="formulario" id="formulario" >
  <input type="text" name="elemento" id="elemento" />
</form>
```

Independientemente del método utilizado para obtener la referencia a un elemento de formulario, cada elemento dispone de las siguientes propiedades útiles para el desarrollo de las aplicaciones:

- `type`: indica el tipo de elemento que se trata. Para los elementos de tipo `<input>` (`text`, `button`, `checkbox`, etc.) coincide con el valor de su atributo `type`. Para las listas desplegables normales (elemento `<select>`) su valor es `select-one`, lo que

permite diferenciarlas de las listas que permiten seleccionar varios elementos a la vez y cuyo tipo es `select-multiple`. Por último, en los elementos de tipo `<textarea>`, el valor de `type` es `textarea`.

- `form`: es una referencia directa al formulario al que pertenece el elemento. Así, para acceder al formulario de un elemento, se puede utilizar `document.getElementById("id_del_elemento").form`
- `name`: obtiene el valor del atributo `name` de XHTML. Solamente se puede leer su valor, por lo que no se puede modificar.
- `value`: permite leer y modificar el valor del atributo `value` de XHTML. Para los campos de texto (`<input type="text">` y `<textarea>`) obtiene el texto que ha escrito el usuario. Para los botones obtiene el texto que se muestra en el botón. Para los elementos *checkbox* y *radiobutton* no es muy útil, como se verá más adelante

Por último, los eventos más utilizados en el manejo de los formularios son los siguientes:

- `onclick`: evento que se produce cuando se pincha con el ratón sobre un elemento. Normalmente se utiliza con cualquiera de los tipos de botones que permite definir XHTML (`<input type="button">`, `<input type="submit">`, `<input type="image">`).
- `onchange`: evento que se produce cuando el usuario cambia el valor de un elemento de texto (`<input type="text">` o `<textarea>`). También se produce cuando el usuario selecciona una opción en una lista desplegable (`<select>`). Sin embargo, el evento sólo se produce si después de realizar el cambio, el usuario *pasa* al siguiente campo del formulario, lo que técnicamente se conoce como que *"el otro campo de formulario ha perdido el foco"*.
- `onfocus`: evento que se produce cuando el usuario selecciona un elemento del formulario.
- `onblur`: evento complementario de `onfocus`, ya que se produce cuando el usuario ha *deseleccionado* un elemento por haber seleccionado otro elemento del formulario. Técnicamente, se dice que el elemento anterior *"ha perdido el foco"*.

7.2. Utilidades básicas para formularios

7.2.1. Obtener el valor de los campos de formulario

La mayoría de técnicas JavaScript relacionadas con los formularios requieren leer y/o modificar el valor de los campos del formulario. Por tanto, a continuación se muestra cómo obtener el valor de los campos de formulario más utilizados.

7.2.1.1. Cuadro de texto y textarea

El valor del texto mostrado por estos elementos se obtiene y se establece directamente mediante la propiedad `value`.

```
<input type="text" id="texto" />
var valor = document.getElementById("texto").value;

<textarea id="parrafo"></textarea>
var valor = document.getElementById("parrafo").value;
```

7.2.1.2. Radiobutton

Cuando se dispone de un grupo de *radiobuttons*, generalmente no se quiere obtener el valor del atributo `value` de alguno de ellos, sino que lo importante es conocer cuál de todos los *radiobuttons* se ha seleccionado. La propiedad `checked` devuelve `true` para el *radiobutton* seleccionado y `false` en cualquier otro caso. Si por ejemplo se dispone del siguiente grupo de *radiobuttons*:

```
<input type="radio" value="si" name="pregunta" id="pregunta_si"/> SI
<input type="radio" value="no" name="pregunta" id="pregunta_no"/> NO
<input type="radio" value="nsnc" name="pregunta" id="pregunta_nsnc"/> NS/NC
```

El siguiente código permite determinar si cada *radiobutton* ha sido seleccionado o no:

```
var elementos = document.getElementsByName("pregunta");

for(var i=0; i<elementos.length; i++) {
  alert(" Elemento: " + elementos[i].value + "\n Seleccionado: " +
elementos[i].checked);
}
```

7.2.1.3. Checkbox

Los elementos de tipo *checkbox* son muy similares a los *radiobutton*, salvo que en este caso se debe comprobar cada *checkbox* de forma independiente del resto. El motivo es que los grupos de *radiobutton* son mutuamente excluyentes y sólo se puede seleccionar uno de ellos cada vez. Por su parte, los *checkbox* se pueden seleccionar de forma independiente respecto de los demás.

Si se dispone de los siguientes *checkbox*:

```
<input type="checkbox" value="condiciones" name="condiciones"
id="condiciones"/> He leído y acepto las condiciones
<input type="checkbox" value="privacidad" name="privacidad" id="privacidad"/>
He leído la política de privacidad
```

Utilizando la propiedad `checked`, es posible comprobar si cada *checkbox* ha sido seleccionado:

```
var elemento = document.getElementById("condiciones");
alert(" Elemento: " + elemento.value + "\n Seleccionado: " + elemento.checked);
```

```
elemento = document.getElementById("privacidad");
alert(" Elemento: " + elemento.value + "\n Seleccionado: " + elemento.checked);
```

7.2.1.4. Select

Las listas desplegables (`<select>`) son los elementos en los que es más difícil obtener su valor. Si se dispone de una lista desplegable como la siguiente:

```
<select id="opciones" name="opciones">
  <option value="1">Primer valor</option>
  <option value="2">Segundo valor</option>
  <option value="3">Tercer valor</option>
  <option value="4">Cuarto valor</option>
</select>
```

En general, lo que se requiere es obtener el valor del atributo `value` de la opción (`<option>`) seleccionada por el usuario. Obtener este valor no es sencillo, ya que se deben realizar una serie de pasos. Además, para obtener el valor seleccionado, deben utilizarse las siguientes propiedades:

- `options`, es un array creado automáticamente por el navegador para cada lista desplegable y que contiene la referencia a todas las opciones de esa lista. De esta forma, la primera opción de una lista se puede obtener mediante `document.getElementById("id_de_la_lista").options[0]`.
- `selectedIndex`, cuando el usuario selecciona una opción, el navegador actualiza automáticamente el valor de esta propiedad, que guarda el índice de la opción seleccionada. El índice hace referencia al array `options` creado automáticamente por el navegador para cada lista.

```
// Obtener la referencia a la lista
var lista = document.getElementById("opciones");

// Obtener el índice de la opción que se ha seleccionado
var indiceSeleccionado = lista.selectedIndex;
// Con el índice y el array "options", obtener la opción seleccionada
var opcionSeleccionada = lista.options[indiceSeleccionado];

// Obtener el valor y el texto de la opción seleccionada
var textoSeleccionado = opcionSeleccionada.text;
var valorSeleccionado = opcionSeleccionada.value;

alert("Opción seleccionada: " + textoSeleccionado + "\n Valor de la opción: " +
valorSeleccionado);
```

Como se ha visto, para obtener el valor del atributo `value` correspondiente a la opción seleccionada por el usuario, es necesario realizar varios pasos. No obstante, normalmente se abrevian todos los pasos necesarios en una única instrucción:

```
var lista = document.getElementById("opciones");

// Obtener el valor de la opción seleccionada
var valorSeleccionado = lista.options[lista.selectedIndex].value;
```

```
// Obtener el texto que muestra la opción seleccionada
var valorSeleccionado = lista.options[lista.selectedIndex].text;
```

Lo más importante es no confundir el valor de la propiedad `selectedIndex` con el valor correspondiente a la propiedad `value` de la opción seleccionada. En el ejemplo anterior, la primera opción tiene un `value` igual a `1`. Sin embargo, si se selecciona esta opción, el valor de `selectedIndex` será `0`, ya que es la primera opción del array `options` (y los arrays empiezan a contar los elementos en el número `0`).

7.2.2. Establecer el foco en un elemento

En programación, cuando un elemento está seleccionado y se puede escribir directamente en el o se puede modificar alguna de sus propiedades, se dice que tiene el foco del programa.

Si un cuadro de texto de un formulario tiene el foco, el usuario puede escribir directamente en el sin necesidad de pinchar previamente con el ratón en el interior del cuadro. Igualmente, si una lista desplegable tiene el foco, el usuario puede seleccionar una opción directamente subiendo y bajando con las flechas del teclado.

Al pulsar repetidamente la tecla `TABULADOR` sobre una página web, los diferentes elementos (enlaces, imágenes, campos de formulario, etc.) van obteniendo el foco del navegador (el elemento seleccionado cada vez suele mostrar un pequeño borde punteado).

Si en una página web el formulario es el elemento más importante, como por ejemplo en una página de búsqueda o en una página con un formulario para registrarse, se considera una buena práctica de usabilidad el asignar automáticamente el foco al primer elemento del formulario cuando se carga la página.

Para asignar el foco a un elemento de XHTML, se utiliza la función `focus()`. El siguiente ejemplo asigna el foco a un elemento de formulario cuyo atributo `id` es igual a `primero`:

```
document.getElementById("primero").focus();

<form id="formulario" action="#">
  <input type="text" id="primero" />
</form>
```

Ampliando el ejemplo anterior, se puede asignar automáticamente el foco del programa al primer elemento del primer formulario de la página, independientemente del `id` del formulario y de los elementos:

```
if(document.forms.length > 0) {
  if(document.forms[0].elements.length > 0) {
    document.forms[0].elements[0].focus();
  }
}
```

El código anterior comprueba que existe al menos un formulario en la página mediante el tamaño del array `forms`. Si su tamaño es mayor que `0`, se utiliza este primer

formulario. Empleando la misma técnica, se comprueba que el formulario tenga al menos un elemento (`if(document.forms[0].elements.length > 0)`). En caso afirmativo, se establece el foco del navegador en el primer elemento del primer formulario (`document.forms[0].elements[0].focus();`).

Para que el ejemplo anterior sea completamente correcto, se debe añadir una comprobación adicional. El campo de formulario que se selecciona no debería ser de tipo `hidden`:

```
if(document.forms.length > 0) {
  for(var i=0; i < document.forms[0].elements.length; i++) {
    var campo = document.forms[0].elements[i];
    if(campo.type != "hidden") {
      campo.focus();
      break;
    }
  }
}
```

7.2.3. Evitar el envío duplicado de un formulario

Uno de los problemas habituales con el uso de formularios web es la posibilidad de que el usuario pulse dos veces seguidas sobre el botón "Enviar". Si la conexión del usuario es demasiado lenta o la respuesta del servidor se hace esperar, el formulario original sigue mostrándose en el navegador y por ese motivo, el usuario tiene la tentación de volver a pinchar sobre el botón de "Enviar".

En la mayoría de los casos, el problema no es grave e incluso es posible controlarlo en el servidor, pero puede complicarse en formularios de aplicaciones importantes como las que implican transacciones económicas.

Por este motivo, una buena práctica en el diseño de aplicaciones web suele ser la de deshabilitar el botón de envío después de la primera pulsación. El siguiente ejemplo muestra el código necesario:

```
<form id="formulario" action="#">
  ...
  <input type="button" value="Enviar" onclick="this.disabled=true;
this.value='Enviando...'; this.form.submit()" />
</form>
```

Cuando se pulsa sobre el botón de envío del formulario, se produce el evento `onclick` sobre el botón y por tanto, se ejecutan las instrucciones JavaScript contenidas en el atributo `onclick`:

1. En primer lugar, se deshabilita el botón mediante la instrucción `this.disabled = true;`. Esta es la única instrucción necesaria si sólo se quiere deshabilitar un botón.
2. A continuación, se cambia el mensaje que muestra el botón. Del original "Enviar" se pasa al más adecuado "Enviando..."

3. Por último, se envía el formulario mediante la función `submit()` en la siguiente instrucción: `this.form.submit()`

El botón del ejemplo anterior está definido mediante un botón de tipo `<input type="button" />`, ya que el código JavaScript mostrado no funciona correctamente con un botón de tipo `<input type="submit" />`. Si se utiliza un botón de tipo `submit`, el botón se deshabilita antes de enviar el formulario y por tanto el formulario acaba sin enviarse.

7.2.4. Limitar el tamaño de caracteres de un textarea

La carencia más importante de los campos de formulario de tipo textarea es la imposibilidad de limitar el máximo número de caracteres que se pueden introducir, de forma similar al atributo `maxlength` de los cuadros de texto normales.

JavaScript permite añadir esta característica de forma muy sencilla. En primer lugar, hay que recordar que con algunos eventos (como `onkeypress`, `onclick` y `onsubmit`) se puede evitar su comportamiento normal si se devuelve el valor `false`.

Evitar el comportamiento normal equivale a modificar completamente el comportamiento habitual del evento. Si por ejemplo se devuelve el valor `false` en el evento `onkeypress`, la tecla pulsada por el usuario no se tiene en cuenta. Si se devuelve `false` en el evento `onclick` de un elemento como un enlace, el navegador no carga la página indicada por el enlace.

Si un evento devuelve el valor `true`, su comportamiento es el habitual:

```
<textarea onkeypress="return true;"></textarea>
```

En el textarea del ejemplo anterior, el usuario puede escribir cualquier carácter, ya que el evento `onkeypress` devuelve `true` y por tanto, su comportamiento es el normal y la tecla pulsada se transforma en un carácter dentro del textarea.

Sin embargo, en el siguiente ejemplo:

```
<textarea onkeypress="return false;"></textarea>
```

Como el valor devuelto por el evento `onkeypress` es igual a `false`, el navegador no ejecuta el comportamiento por defecto del evento, es decir, la tecla presionada no se transforma en ningún carácter dentro del textarea. No importa las veces que se pulsen las teclas y no importa la tecla pulsada, ese textarea no permitirá escribir ningún carácter.

Aprovechando esta característica, es sencillo limitar el número de caracteres que se pueden escribir en un elemento de tipo textarea: se comprueba si se ha llegado al máximo número de caracteres permitido y en caso afirmativo se evita el comportamiento habitual del evento y por tanto, los caracteres adicionales no se añaden al textarea:

```
function limita(maximoCaracteres) {
  var elemento = document.getElementById("texto");
  if(elemento.value.length >= maximoCaracteres ) {
    return false;
  }
  else {
    return true;
  }
}

<textarea id="texto" onkeypress="return limita(100);"></textarea>
```

En el ejemplo anterior, con cada tecla pulsada se compara el número total de caracteres del textarea con el máximo número de caracteres permitido. Si el número de caracteres es igual o mayor que el límite, se devuelve el valor `false` y por tanto, se evita el comportamiento por defecto de `onkeypress` y la tecla no se añade.

Ejercicio 17

Mejorar el ejemplo anterior indicando en todo momento al usuario el número de caracteres que aún puede escribir. Además, se debe permitir pulsar las teclas `Backspace`, `Supr.` y las flechas horizontales cuando se haya llegado al máximo número de caracteres.

7.2.5. Restringir los caracteres permitidos en un cuadro de texto

En ocasiones, puede ser útil bloquear algunos caracteres determinados en un cuadro de texto. Si por ejemplo un cuadro de texto espera que se introduzca un número, puede ser interesante no permitir al usuario introducir ningún carácter que no sea numérico.

Igualmente, en algunos casos puede ser útil impedir que el usuario introduzca números en un cuadro de texto. Utilizando el evento `onkeypress` y unas cuantas sentencias JavaScript, el problema se resuelve fácilmente:

```
function permite(elEvento, permitidos) {
  // Variables que definen los caracteres permitidos
  var numeros = "0123456789";
  var caracteres = " abcdefghijklmnñopqrstuvwxyzABCDEFGHIJKLMNÑOPQRSTUVWXYZ";
  var numeros_caracteres = numeros + caracteres;
  var teclas_especiales = [8, 37, 39, 46];
  // 8 = BackSpace, 46 = Supr, 37 = flecha izquierda, 39 = flecha derecha


  // Seleccionar los caracteres a partir del parámetro de la función
  switch(permitidos) {
    case 'num':
      permitidos = numeros;
      break;
    case 'car':
      permitidos = caracteres;
      break;
    case 'num_car':
      permitidos = numeros_caracteres;
      break;
```

```
  }

  // Obtener la tecla pulsada
  var evento = elEvento || window.event;
  var codigoCaracter = evento.charCode || evento.keyCode;
  var caracter = String.fromCharCode(codigoCaracter);

  // Comprobar si la tecla pulsada es alguna de las teclas especiales
  // (teclas de borrado y flechas horizontales)
  var tecla_especial = false;
  for(var i in teclas_especiales) {
    if(codigoCaracter == teclas_especiales[i]) {
      tecla_especial = true;
      break;
    }
  }

  // Comprobar si la tecla pulsada se encuentra en los caracteres permitidos
  // o si es una tecla especial
  return permitidos.indexOf(caracter) != -1 || tecla_especial;
}

// Sólo números
<input type="text" id="texto" onkeypress="return permite(event, 'num')" />

// Sólo letras
<input type="text" id="texto" onkeypress="return permite(event, 'car')" />

// Sólo letras o números
<input type="text" id="texto" onkeypress="return permite(event, 'num_car')" />
```

El funcionamiento del script anterior se basa en permitir o impedir el comportamiento habitual del evento `onkeypress`. Cuando se pulsa una tecla, se comprueba si el carácter de esa tecla se encuentra dentro de los caracteres permitidos para ese elemento `<input>`.

Si el carácter se encuentra dentro de los caracteres permitidos, se devuelve `true` y por tanto el comportamiento de `onkeypress` es el habitual y la tecla se escribe. Si el carácter no se encuentra dentro de los caracteres permitidos, se devuelve `false` y por tanto se impide el comportamiento normal de `onkeypress` y la tecla no llega a escribirse en el input.

Además, el script anterior siempre permite la pulsación de algunas teclas *especiales*. En concreto, las teclas `BackSpace` y `Supr` para borrar caracteres y las teclas `Flecha Izquierda` y `Flecha Derecha` para moverse en el cuadro de texto siempre se puden pulsar independientemente del tipo de caracteres permitidos.

7.3. Validación

La principal utilidad de JavaScript en el manejo de los formularios es la validación de los datos introducidos por los usuarios. Antes de enviar un formulario al servidor, se

recomienda validar mediante JavaScript los datos insertados por el usuario. De esta forma, si el usuario ha cometido algún error al rellenar el formulario, se le puede notificar de forma instantánea, sin necesidad de esperar la respuesta del servidor.

Notificar los errores de forma inmediata mediante JavaScript mejora la satisfacción del usuario con la aplicación (lo que técnicamente se conoce como *"mejorar la experiencia de usuario"*) y ayuda a reducir la carga de procesamiento en el servidor.

Normalmente, la validación de un formulario consiste en llamar a una función de validación cuando el usuario pulsa sobre el botón de envío del formulario. En esta función, se comprueban si los valores que ha introducido el usuario cumplen las restricciones impuestas por la aplicación.

Aunque existen tantas posibles comprobaciones como elementos de formulario diferentes, algunas comprobaciones son muy habituales: que se rellene un campo obligatorio, que se seleccione el valor de una lista desplegable, que la dirección de email indicada sea correcta, que la fecha introducida sea lógica, que se haya introducido un número donde así se requiere, etc.

A continuación se muestra el código JavaScript básico necesario para incorporar la validación a un formulario:

```
<form action="" method="" id="" name="" onsubmit="return validacion()">
  ...
</form>
```

Y el esquema de la función `validacion()` es el siguiente:

```
function validacion() {
  if (condicion que debe cumplir el primer campo del formulario) {
    // Si no se cumple la condicion...
    alert('[ERROR] El campo debe tener un valor de...');
    return false;
  }
  else if (condicion que debe cumplir el segundo campo del formulario) {
    // Si no se cumple la condicion...
    alert('[ERROR] El campo debe tener un valor de...');
    return false;
  }
  ...
  else if (condicion que debe cumplir el último campo del formulario) {
    // Si no se cumple la condicion...
    alert('[ERROR] El campo debe tener un valor de...');
    return false;
  }

  // Si el script ha llegado a este punto, todas las condiciones
  // se han cumplido, por lo que se devuelve el valor true
  return true;
}
```

El funcionamiento de esta técnica de validación se basa en el comportamiento del evento `onsubmit` de JavaScript. Al igual que otros eventos como `onclick` y `onkeypress`, el evento `'onsubmit` varía su comportamiento en función del valor que se devuelve.

Así, si el evento `onsubmit` devuelve el valor `true`, el formulario se envía como lo haría normalmente. Sin embargo, si el evento `onsubmit` devuelve el valor `false`, el formulario no se envía. La clave de esta técnica consiste en comprobar todos y cada uno de los elementos del formulario. En cuando se encuentra un elemento incorrecto, se devuelve el valor `false`. Si no se encuentra ningún error, se devuelve el valor `true`.

Por lo tanto, en primer lugar se define el evento `onsubmit` del formulario como:

```
onsubmit="return validacion()"
```

Como el código JavaScript devuelve el valor resultante de la función `validacion()`, el formulario solamente se enviará al servidor si esa función devuelve `true`. En el caso de que la función `validacion()` devuelva `false`, el formulario permanecerá sin enviarse.

Dentro de la función `validacion()` se comprueban todas las condiciones impuestas por la aplicación. Cuando no se cumple una condición, se devuelve `false` y por tanto el formulario no se envía. Si se llega al final de la función, todas las condiciones se han cumplido correctamente, por lo que se devuelve `true` y el formulario se envía.

La notificación de los errores cometidos depende del diseño de cada aplicación. En el código del ejemplo anterior simplemente se muestran mensajes mediante la función `alert()` indicando el error producido. Las aplicaciones web mejor diseñadas muestran cada mensaje de error al lado del elemento de formulario correspondiente y también suelen mostrar un mensaje principal indicando que el formulario contiene errores.

Una vez definido el esquema de la función `validacion()`, se debe añadir a esta función el código correspondiente a todas las comprobaciones que se realizan sobre los elementos del formulario. A continuación, se muestran algunas de las validaciones más habituales de los campos de formulario.

7.3.1. Validar un campo de texto obligatorio

Se trata de forzar al usuario a introducir un valor en un cuadro de texto o textarea en los que sea obligatorio. La condición en JavaScript se puede indicar como:

```
valor = document.getElementById("campo").value;
if( valor == null || valor.length == 0 || /^\s+$/.test(valor) ) {
  return false;
}
```

Para que se de por completado un campo de texto obligatorio, se comprueba que el valor introducido sea válido, que el número de caracteres introducido sea mayor que cero y que no se hayan introducido sólo espacios en blanco.

La palabra reservada `null` es un valor especial que se utiliza para indicar *"ningún valor"*. Si el valor de una variable es `null`, la variable no contiene ningún valor de tipo objeto, array, numérico, cadena de texto o booleano.

La segunda parte de la condición obliga a que el texto introducido tenga una longitud superior a cero caracteres, esto es, que no sea un texto vacío.

Por último, la tercera parte de la condición (`/^\s+$/.test(valor)`) obliga a que el valor introducido por el usuario no sólo esté formado por espacios en blanco. Esta comprobación se basa en el uso de *"expresiones regulares"*, un recurso habitual en cualquier lenguaje de programación pero que por su gran complejidad no se van a estudiar. Por lo tanto, sólo es necesario copiar literalmente esta condición, poniendo especial cuidado en no modificar ningún carácter de la expresión.

7.3.2. Validar un campo de texto con valores numéricos

Se trata de obligar al usuario a introducir un valor numérico en un cuadro de texto. La condición JavaScript consiste en:

```
valor = document.getElementById("campo").value;
if( isNaN(valor) ) {
  return false;
}
```

Si el contenido de la variable `valor` no es un número válido, no se cumple la condición. La ventaja de utilizar la función interna `isNaN()` es que simplifica las comprobaciones, ya que JavaScript se encarga de tener en cuenta los decimales, signos, etc.

A continuación se muestran algunos resultados de la función `isNaN()`:

```
isNaN(3);          // false
isNaN("3");        // false
isNaN(3.3545);     // false
isNaN(32323.345);  // false
isNaN(+23.2);      // false
isNaN("-23.2");    // false
isNaN("23a");      // true
isNaN("23.43.54"); // true
```

7.3.3. Validar que se ha seleccionado una opción de una lista

Se trata de obligar al usuario a seleccionar un elemento de una lista desplegable. El siguiente código JavaScript permite conseguirlo:

```
indice = document.getElementById("opciones").selectedIndex;
if( indice == null || indice == 0 ) {
  return false;
}

<select id="opciones" name="opciones">
  <option value="">- Selecciona un valor -</option>
  <option value="1">Primer valor</option>
  <option value="2">Segundo valor</option>
```

```
  <option value="3">Tercer valor</option>
</select>
```

A partir de la propiedad `selectedIndex`, se comprueba si el índice de la opción seleccionada es válido y además es distinto de cero. La primera opción de la lista (`- Selecciona un valor –`) no es válida, por lo que no se permite el valor `0` para esta propiedad `selectedIndex`.

7.3.4. Validar una dirección de email

Se trata de obligar al usuario a introducir una dirección de email con un formato válido. Por tanto, lo que se comprueba es que la dirección *parezca* válida, ya que no se comprueba si se trata de una cuenta de correo electrónico real y operativa. La condición JavaScript consiste en:

```
valor = document.getElementById("campo").value;
if( !(/\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)/.test(valor)) ) {
  return false;
}
```

La comprobación se realiza nuevamente mediante las expresiones regulares, ya que las direcciones de correo electrónico válidas pueden ser muy diferentes. Por otra parte, como el estándar que define el formato de las direcciones de correo electrónico es muy complejo, la expresión regular anterior es una simplificación. Aunque esta regla valida la mayoría de direcciones de correo electrónico utilizadas por los usuarios, no soporta todos los diferentes formatos válidos de email.

7.3.5. Validar una fecha

Las fechas suelen ser los campos de formulario más complicados de validar por la multitud de formas diferentes en las que se pueden introducir. El siguiente código asume que de alguna forma se ha obtenido el año, el mes y el día introducidos por el usuario:

```
var ano = document.getElementById("ano").value;
var mes = document.getElementById("mes").value;
var dia = document.getElementById("dia").value;

valor = new Date(ano, mes, dia);

if( !isNaN(valor) ) {
  return false;
}
```

La función `Date(ano, mes, dia)` es una función interna de JavaScript que permite construir fechas a partir del año, el mes y el día de la fecha. Es muy importante tener en cuenta que el número de mes se indica de `0` a `11`, siendo `0` el mes de Enero y `11` el mes de Diciembre. Los días del mes siguen una numeración diferente, ya que el mínimo permitido es `1` y el máximo `31`.

La validación consiste en intentar construir una fecha con los datos proporcionados por el usuario. Si los datos del usuario no son correctos, la fecha no se puede construir correctamente y por tanto la validación del formulario no será correcta.

7.3.6. Validar un número de DNI

Se trata de comprobar que el número proporcionado por el usuario se corresponde con un número válido de Documento Nacional de Identidad o DNI. Aunque para cada país o región los requisitos del documento de identidad de las personas pueden variar, a continuación se muestra un ejemplo genérico fácilmente adaptable. La validación no sólo debe comprobar que el número esté formado por ocho cifras y una letra, sino que también es necesario comprobar que la letra indicada es correcta para el número introducido:

```
valor = document.getElementById("campo").value;
var letras = ['T', 'R', 'W', 'A', 'G', 'M', 'Y', 'F', 'P', 'D', 'X', 'B', 'N',
'J', 'Z', 'S', 'Q', 'V', 'H', 'L', 'C', 'K', 'E', 'T'];

if( !(/^\d{8}[A-Z]$/.test(valor)) ) {
  return false;
}

if(valor.charAt(8) != letras[(valor.substring(0, 8))%23]) {
  return false;
}
```

La primera comprobación asegura que el formato del número introducido es el correcto, es decir, que está formado por 8 números seguidos y una letra. Si la letra está al principio de los números, la comprobación sería `/^[A-Z]\d{8}$/`. Si en vez de ocho números y una letra, se requieren diez números y dos letras, la comprobación sería `/^\d{10}[A-Z]{2}$/` y así sucesivamente.

La segunda comprobación aplica el algoritmo de cálculo de la letra del DNI y la compara con la letra proporcionada por el usuario. El algoritmo de cada documento de identificación es diferente, por lo que esta parte de la validación se debe adaptar convenientemente.

7.3.7. Validar un número de teléfono

Los números de teléfono pueden ser indicados de formas muy diferentes: con prefijo nacional, con prefijo internacional, agrupado por pares, separando los números con guiones, etc.

El siguiente script considera que un número de teléfono está formado por nueve dígitos consecutivos y sin espacios ni guiones entre las cifras:

```
valor = document.getElementById("campo").value;
if( !(/^\d{9}$/.test(valor)) ) {
  return false;
}
```

Una vez más, la condición de JavaScript se basa en el uso de expresiones regulares, que comprueban si el valor indicado es una sucesión de nueve números consecutivos. A continuación se muestran otras expresiones regulares que se pueden utilizar para otros formatos de número de teléfono:

Número	Expresión regular	Formato
900900900	/^\d{9}$/	9 cifras seguidas
900-900-900	/^\d{3}-\d{3}-\d{3}$/	9 cifras agrupadas de 3 en 3 y separadas por guiones
900 900900	/^\d{3}\s\d{6}$/	9 cifras, las 3 primeras separadas por un espacio
900 90 09 00	/^\d{3}\s\d{2}\s\d{2}\s\d{2}$/	9 cifras las 3 primeras separadas por un espacio las siguientes agrupadas de 2 en 2
(900) 900900	/^\(\d{3}\)\s\d{6}$/	9 cifras las 3 primeras encerradas por paréntesis y un espacio de separación respecto del resto
+34 900900900	/^\+\d{2,3}\s\d{9}$/	Prefijo internacional (+ seguido de 2 o 3 cifras) espacio en blanco y 9 cifras consecutivas

7.3.8. Validar que un checkbox ha sido seleccionado

Si un elemento de tipo *checkbox* se debe seleccionar de forma obligatoria, JavaScript permite comprobarlo de forma muy sencilla:

```
elemento = document.getElementById("campo");
if( !elemento.checked ) {
  return false;
}
```

Si se trata de comprobar que todos los *checkbox* del formulario han sido seleccionados, es más fácil utilizar un bucle:

```
formulario = document.getElementById("formulario");
for(var i=0; i<formulario.elements.length; i++) {
  var elemento = formulario.elements[i];
  if(elemento.type == "checkbox") {
    if(!elemento.checked) {
      return false;
    }
  }
}
```

7.3.9. Validar que un radiobutton ha sido seleccionado

Aunque se trata de un caso similar al de los *checkbox*, la validación de los *radiobutton* presenta una diferencia importante: en general, la comprobación que se realiza es que el

usuario haya seleccionado algún *radiobutton* de los que forman un determinado grupo. Mediante JavaScript, es sencillo determinar si se ha seleccionado algún *radiobutton* de un grupo:

```
opciones = document.getElementsByName("opciones");

var seleccionado = false;
for(var i=0; i<opciones.length; i++) {
  if(opciones[i].checked) {
    seleccionado = true;
    break;
  }
}

if(!seleccionado) {
  return false;
}
```

El anterior ejemplo recorre todos los *radiobutton* que forman un grupo y comprueba elemento por elemento si ha sido seleccionado. Cuando se encuentra el primer *radiobutton* seleccionado, se sale del bucle y se indica que al menos uno ha sido seleccionado.

Capítulo 8. Otras utilidades

Además de los formularios y de todas las funciones y utilidades de JavaScript que se han visto, existen muchas otras utilidades que es necesario conocer para desarrollar aplicaciones completas. Como no es posible estudiar todas las herramientas que se pueden crear con JavaScript, a continuación se muestran algunas de las utilidades básicas más comunes.

8.1. Relojes, contadores e intervalos de tiempo

En ocasiones, algunas páginas web muestran un reloj con la hora actual. Si el reloj debe actualizarse cada segundo, no se puede mostrar la hora directamente en la página HTML generada por el servidor. En este caso, aunque existen alternativas realizadas con Java y con Flash, la forma más sencilla de hacerlo es mostrar la hora del ordenador del usuario mediante JavaScript.

Para crear y mostrar un reloj con JavaScript, se debe utilizar el objeto interno `Date()` para crear fechas/horas y las utilidades que permiten definir contadores, para actualizar el reloj cada segundo.

El objeto `Date()` es una utilidad que proporciona JavaScript para crear fechas y horas. Una vez creado un objeto de tipo fecha, es posible manipularlo para obtener información o realizar cálculos con las fechas. Para obtener la fecha y hora actuales, solamente es necesario crear un objeto `Date()` sin pasar ningún parámetro:

```
var fechaHora = new Date();
```

Utilizando el código anterior, se puede construir un reloj muy básico que no actualiza su contenido:

```
var fechaHora = new Date();
document.getElementById("reloj").innerHTML = fechaHora;

<div id="reloj" />
```

Cuando se carga la página, el ejemplo anterior mostraría un texto parecido al siguiente en el `<div>` reservado para el reloj:

```
Thu Aug 02 2007 19:35:19 GMT+0200 (Hora de verano romance)
```

Este primer reloj construido presenta muchas diferencias respecto al reloj que se quiere construir. En primer lugar, muestra mucha más información de la necesaria. Además, su valor no se actualiza cada segundo, por lo que no es un reloj muy práctico.

El objeto `Date()` proporciona unas funciones muy útiles para obtener información sobre la fecha y la hora. Concretamente, existen funciones que obtienen directamente la hora, los minutos y los segundos:

```
var fechaHora = new Date();
var horas = fechaHora.getHours();
var minutos = fechaHora.getMinutes();
var segundos = fechaHora.getSeconds();
document.getElementById("reloj").innerHTML = horas+':'+minutos+':'+segundos;

<div id="reloj" />
```

Utilizando las funciones `getHours()`, `getMinutes()` y `getSeconds()` del objeto `Date`, el reloj puede mostrar solamente la información de la hora. El resultado del ejemplo anterior sería un reloj como el siguiente:

```
20:9:21
```

Si la hora, minuto o segundo son menores que `10`, JavaScript no añade el `0` por delante, por lo que el resultado no es del todo satisfactorio. El siguiente código soluciona este problema añadiendo un `0` cuando sea necesario:

```
var fechaHora = new Date();
var horas = fechaHora.getHours();
var minutos = fechaHora.getMinutes();
var segundos = fechaHora.getSeconds();

if(horas < 10) { horas = '0' + horas; }
if(minutos < 10) { minutos = '0' + minutos; }
if(segundos < 10) { segundos = '0' + segundos; }

document.getElementById("reloj").innerHTML = horas+':'+minutos+':'+segundos;

<div id="reloj" />
```

Ahora el reloj muestra correctamente la hora:

```
20:14:03
```

Para completar el reloj, sólo falta que se actualice su valor cada segundo. Para conseguirlo, se deben utilizar unas funciones especiales de JavaScript que permiten ejecutar determinadas instrucciones cuando ha transcurrido un determinado espacio de tiempo.

La función `setTimeout()` permite ejecutar una función una vez que haya transcurrido un periodo de tiempo indicado. La definición de la función es:

```
setTimeout(nombreFuncion, milisegundos);
```

La función que se va a ejecutar se debe indicar mediante su nombre sin paréntesis y el tiempo que debe transcurrir hasta que se ejecute se indica en milisegundos. De esta forma, si se crea una función encargada de mostrar la hora del reloj y se denomina `muestraReloj()`, se puede indicar que se ejecute dentro de 1 segundo mediante el siguiente código:

```
function muestraReloj() {
  var fechaHora = new Date();
  var horas = fechaHora.getHours();
  var minutos = fechaHora.getMinutes();
```

```
  var segundos = fechaHora.getSeconds();

  if(horas < 10) { horas = '0' + horas; }
  if(minutos < 10) { minutos = '0' + minutos; }
  if(segundos < 10) { segundos = '0' + segundos; }

  document.getElementById("reloj").innerHTML = horas+':'+minutos+':'+segundos;
}

setTimeout(muestraReloj, 1000);

<div id="reloj" />
```

No obstante, el código anterior simplemente muestra el contenido del reloj 1 segundo después de que se cargue la página, por lo que no es muy útil. Para ejecutar una función de forma periódica, se utiliza una función de JavaScript muy similar a `setTimeout()` que se denomina `setInterval()`. Su definición es:

```
setInterval(nombreFuncion, milisegundos);
```

La definición de esta función es idéntica a la función `setTimeout()`, salvo que en este caso, la función programada se ejecuta infinitas veces de forma periódica con un lapso de tiempo entre ejecuciones de tantos milisegundos como se hayan establecido.

Así, para construir el reloj completo, se establece una ejecución periódica de la función `muestraReloj()` cada segundo:

```
function muestraReloj() {
  var fechaHora = new Date();
  var horas = fechaHora.getHours();
  var minutos = fechaHora.getMinutes();
  var segundos = fechaHora.getSeconds();

  if(horas < 10) { horas = '0' + horas; }
  if(minutos < 10) { minutos = '0' + minutos; }
  if(segundos < 10) { segundos = '0' + segundos; }

  document.getElementById("reloj").innerHTML = horas+':'+minutos+':'+segundos;
}

window.onload = function() {
  setInterval(muestraReloj, 1000);
}

<div id="reloj" />
```

Empleando el objeto `Date` y sus funciones, es posible construir *"cuentras atrás"*, es decir, relojes que muestran el tiempo que falta hasta que se produzca un evento. Además, las funciones `setTimeout()` y `setInterval()` pueden resultar muy útiles en otras técnicas de programación.

8.2. Calendario

Cuando una aplicación muestra un formulario para que el usuario inserte sus datos, la información más complicada de obtener siempre suelen ser los números de teléfono y las fechas. El motivo es que existen muchas formas diferentes de indicar estos datos. Las fechas por ejemplo se pueden indicar como `dd/mm/aa`, `dd/mm/aaaa`, `aaaa/mm/dd`, `dd-mm-aa`, `dd mm aaaa`, etc.

Los métodos más utilizados para que el usuario introduzca la información relativa a una fecha suelen ser un cuadro de texto en el que tiene que insertar la fecha completa (indicándole el formato que debe seguir) o un cuadro de texto para el día, una lista desplegable para el mes y otro cuadro de texto para el año.

En cualquier caso, para el usuario suele ser más cómodo que la aplicación incluya un calendario en el que pueda indicar la fecha pinchando sobre el día elegido:

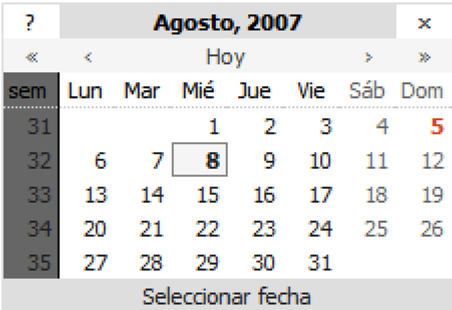


Figura 8.1. Aspecto de un calendario creado con JavaScript

Además, este método es menos propenso a cometer errores, ya que si el calendario está bien construido, no es posible introducir fechas inválidas y tampoco es posible insertar las fechas en un formato incorrecto.

No obstante, realizar un calendario completo en JavaScript no es una tarea trivial, por lo que no se va a estudiar su desarrollo completo. Afortunadamente, existen scripts gratuitos para mostrar calendarios y que permiten su uso libre incluso en aplicaciones comerciales.

De entre todos los calendarios disponibles, uno de los más completos es el desarrollado por la empresa DynArch, que se puede acceder desde su página web oficial: http://www.dynarch.com/projects/calendar/ y que se puede descargar gratuitamente desde http://sourceforge.net/projects/jscalendar/ El archivo descargado incluye todos los scripts necesarios, su documentación, ejemplos de uso, diferentes estilos CSS para el calendario, etc.

A continuación se indican los pasos necesarios para incluir un calendario básico en cualquier página web:

1) Enlazar los archivos JavaScript y CSS requeridos:

Se descomprime el archivo descargado, se guardan los archivos JavaScript y CSS en el sitio adecuado (en este ejemplo se supone que los archivos JavaScript se guardan en el

directorio `js/` y los archivos CSS en el directorio `css/`) y se enlazan directamente desde la cabecera de la página HTML.

```
<head>
...
<style type="text/css">@import url("css/calendar-estilo.css");</style>
<script type="text/javascript" src="js/calendar.js" />
<script type="text/javascript" src="js/calendar-es.js" />
<script type="text/javascript" src="js/calendar-setup.js" />
...
</head>
```

El calendario incluye traducciones a más de 40 idiomas, entre los que se encuentra el español. Para mostrar el calendario en un determinado idioma, tan solo es necesario enlazar el archivo del idioma correspondiente. Las traducciones se encuentran en el directorio `lang` y su formato es `calendar-XX.js`, donde `XX` es el código del idioma.

2) Añadir el código XHTML necesario para el elemento que va a mostrar el calendario:

```
<p>Introduce la fecha pulsando sobre la imagen del calendario</p>
<input type="text" name="date" id="fecha" readonly="readonly" />
<img src="calendario.png" id="selector" />
```

En este caso, el calendario está formado por dos elementos:

- Un cuadro de texto llamado `fecha` y que almacena el valor introducido por el usuario. Como se le ha asignado un atributo `readonly="readonly"`, el usuario no puede modificar su valor.
- Una pequeña imagen o icono que se utiliza para activar el calendario. Cuando el usuario pincha con el ratón sobre la imagen, se muestra el calendario de JavaScript.

3) Configurar el calendario:

```
<script type="text/javascript">
window.onload = function() {
  Calendar.setup({
    inputField: "fecha",
    ifFormat:   "%d / %m / %Y",
    button:     "selector"
  });
}
</script>
```

Una vez enlazados los archivos del calendario y preparado el código XHTML, es necesario inicializar y configurar el calendario. La configuración del calendario consiste en establecer el valor de alguna de sus propiedades. Las propiedades básicas imprescindibles son:

- `inputField`: se trata del atributo `id` del elemento en el que se mostrará la fecha seleccionada, en este ejemplo sería `fecha`.

- `ifFormat`: formato en el que se mostrará la fecha una vez seleccionada (en este caso se ha optado por el formato `dd / mm / aaaa`).
- `button`: atributo `id` del elemento que se pulsa (botón, imagen, enlace) para mostrar el calendario de selección de fecha. En este ejemplo, el `id` de la imagen es `selector`.

Después de esta configuración básica, el calendario ya puede utilizarse en la aplicación:

1) Aspecto por defecto del selector de calendario:

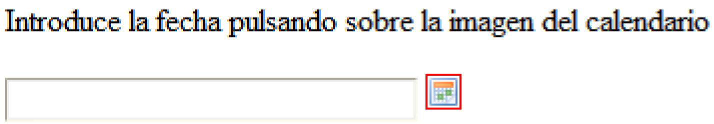


Figura 8.2. Elementos XHTML del calendario JavaScript: cuadro de texto e icono

2) Al pinchar una vez sobre la imagen del calendario:

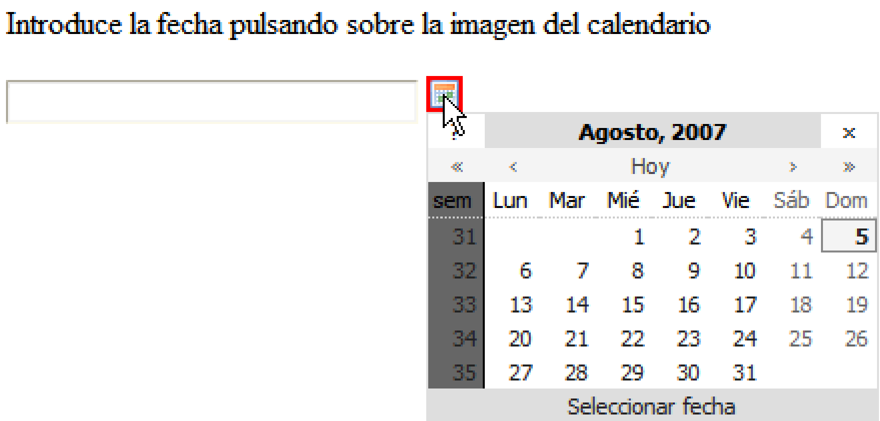


Figura 8.3. Aspecto inicial del calendario JavaScript cuando se abre por primera vez

Se muestra el calendario con el aspecto e idioma establecidos y aparece resaltada la fecha del día actual.

3) Se selecciona la fecha deseada:

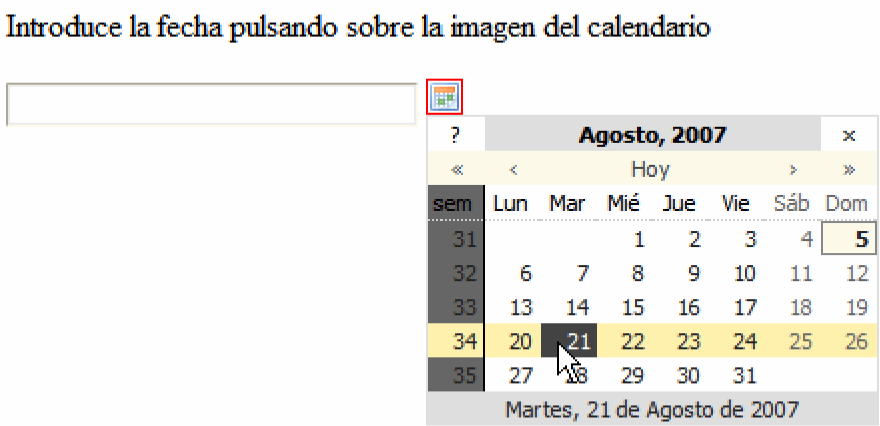


Figura 8.4. Seleccionando una fecha en el calendario JavaScript

4) Después de pinchar con el ratón sobre la fecha deseada:

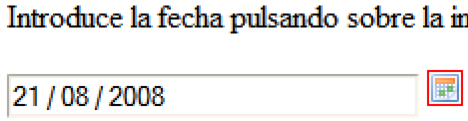


Figura 8.5. El cuadro de texto muestra la fecha establecida por el usuario con el calendario JavaScript

Al pulsar con el ratón sobre la fecha deseada, el calendario se cierra automáticamente y el cuadro de texto muestra la fecha seleccionada con el formato indicado en la configuración del calendario. Si se vuelve a abrir el calendario para seleccionar otra fecha, se mostrará resaltada la fecha del día seleccionado y no la fecha del día actual.

Ejercicio 18

Mejorar el calendario creado añadiendo las opciones necesarias para que muestre el siguiente aspecto:



Figura 8.6. Aspecto mejorado del calendario JavaScript

1. Que no se muestre el número de la semana en el calendario (pista: `weekNumbers`)
2. Modificar el formato en el que se muestra la fecha seleccionada. El formato original es `21 / 08 / 2007` (indicado como `%d / %m / %Y`). El formato deseado es `Martes, 21 de Agosto de 2007` (pistas: `%A`, `%B`)
3. El nuevo formato de fecha es mucho más agradable para los usuarios, pero más incómodo para los scripts que tienen que manejarlo. Afortunadamente, el calendario dispone de la posibilidad de guardar dos valores: un valor para mostrar a los usuarios y otro valor para que lo procesen los scripts. Cuando el usuario seleccione una fecha, la fecha con el formato original se debe almacenar en un campo oculto de formulario y el otro formato más largo debe mostrar al usuario en un elemento de tipo `<span>` (pistas: `displayArea`, `daFormat`)

8.3. Tooltip

Los *tooltips* son pequeños recuadros de información que aparecen al posicionar el ratón sobre un elemento. Normalmente se utilizan para ofrecer información adicional sobre el elemento seleccionado o para mostrar al usuario pequeños mensajes de ayuda. Los *tooltips* son habituales en varios elementos de los sistemas operativos:

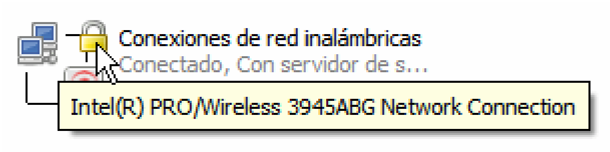


Figura 8.7. Aspecto de un tooltip típico en el Sistema Operativo

Realizar un *tooltip* completo mediante JavaScript es una tarea muy compleja, sobre todo por la dificultad de mostrar el mensaje correctamente en función de la posición del ratón. Afortunadamente, existen scripts que ya están preparados para generar cualquier tipo de *tooltip*. La librería que se va a utilizar se denomina overLIB, y se puede descargar desde su página web principal: http://www.bosrup.com/web/overlib/

La descarga incluye todos los scripts necesarios para el funcionamiento del sistema de *tooltips*, pero no su documentación, que se puede consultar en: http://www.bosrup.com/web/overlib/?Documentation. La referencia de todos los comandos disponibles se puede consultar en: http://www.bosrup.com/web/overlib/?Command_Reference

A continuación se indican los pasos necesarios para incluir un *tooltip* básico en cualquier página web:

1) Enlazar los archivos JavaScript requeridos:

```
<script type="text/javascript" src="js/overlib.js"><!-- overLIB (c) Erik Bosrup
--></script>
```

Se descomprime el archivo descargado, se guarda el archivo JavaScript en el sitio adecuado (en este ejemplo se supone que los archivos JavaScript se guardan en el directorio `js/`) y se enlaza directamente desde la cabecera de la página HTML. Los *tooltips* sólo requieren que se enlace un único archivo JavaScript (`overlib.js`). El comentario que incluye el código XHTML es el aviso de copyright de la librería, que es obligatorio incluirlo para poder usarla.

2) Definir el texto que activa el *tooltip* y su contenido:

```
<p>Lorem ipsum dolor sit amet, <a href="#" onmouseover="return overlib('Prueba
de un tooltip básico y muy sencillo.');" onmouseout="return nd();">consectetuer
adipiscing elit</a>. Etiam eget metus. Proin varius auctor tortor. Cras augue
neque, porta vitae, vestibulum nec, pulvinar id, nibh. Fusce in arcu. Duis
vehicula nonummy orci.</p>
```

Los tooltip se incluyen como enlaces de tipo `<a>` para los que se definen los eventos `onmouseover` y `onmouseout`. Cuando el ratón se pasa por encima del enlace anterior, se muestra el *tooltip* con el aspecto por defecto:

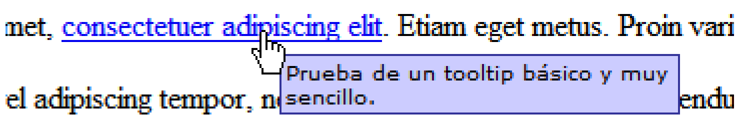


Figura 8.8. Aspecto por defecto del tooltip creado con la librería overLIB

La librería overLIB permite configurar completamente el aspecto y comportamiento de cada *tooltip*. Las opciones se indican en cada *tooltip* y se incluyen como parámetros al final de la llamada a la función `overlib()`:

```
<a href="#" onmouseover="return overlib('Prueba de un tooltip básico y muy
sencillo.', CENTER);" onmouseout="return nd();">consectetuer adipiscing elit</a>
```

El parámetro `CENTER` indica que el *tooltip* se debe mostrar centrado respecto de la posición del ratón:



Figura 8.9. Centrando el tooltip respecto de la posición del ratón

Otra opción que se puede controlar es la anchura del *tooltip*. Por defecto, su anchura es de `200px`, pero la opción `WIDTH` permite modificarla:

```
<a href="#" onmouseover="return overlib('Prueba de un tooltip básico y muy
sencillo.', CENTER, WIDTH, 120);" onmouseout="return nd();">consectetuer
adipiscing elit</a>
```

El valor de la nueva anchura se indica en el parámetro que va detrás de `WIDTH`. El nuevo aspecto del *tooltip* es el siguiente:

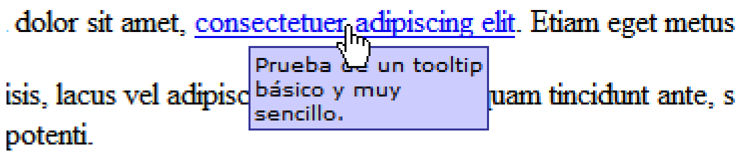


Figura 8.10. Definiendo la anchura que muestra el tooltip

El uso de los parámetros de configuración en la propia llamada a la función que muestra los *tooltips* no es recomendable cuando el número de parámetros empieza a ser muy numeroso. Afortunadamente, overLIB permite realizar una única configuración que se utilizará para todos los *tooltips* de la página.

La configuración global consiste en llamar a la función `overlib_pagedefaults()` con todos los parámetros de configuración deseados. Por tanto, el código JavaScript necesario para realizar los cambios configurados hasta el momento sería:

```
<script type="text/javascript" src="js/overlib.js"><!-- overLIB (c) Erik Bosrup
--></script>
<script type="text/javascript">
overlib_pagedefaults(CENTER, WIDTH, 120);
</script>

<p>Lorem ipsum dolor sit amet, <a href="#" onmouseover="return overlib('Prueba
de un tooltip básico y muy sencillo.');" onmouseout="return nd();">consectetuer
adipiscing elit</a>. Etiam eget metus. Proin varius auctor tortor. Cras augue
neque, porta vitae, vestibulum nec, pulvinar id, nibh. Fusce in arcu. Duis
vehicula nonummy orci.</p>
```

Utilizando esta configuración global, se va a modificar por último el aspecto del *tooltip* para hacerlo más parecido a los tooltips de los sistemas operativos:

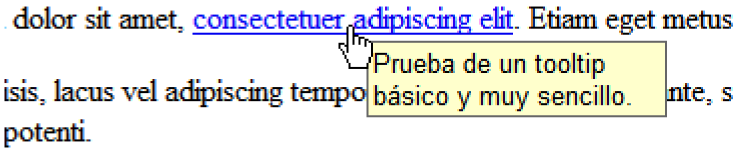


Figura 8.11. Tooltip creado con la librería overLIB y personalizado para que parezca un tooltip de Sistema Operativo

En el ejemplo anterior se ha modificado el tamaño y tipo de letra y el color de fondo del *tooltip* mediante la siguiente configuración:

```
overlib_pagedefaults(WIDTH,150,FGCOLOR,'#ffffcc',BGCOLOR,'#666666',TEXTFONT,"Arial,
Helvetica, Verdana",TEXTSIZE,".8em");
```

Ejercicio 19

Mejorar el *tooltip* propuesto añadiendo las siguientes características:

1. Que el *tooltip* no se muestre instantáneamente, sino que transcurra un cuarto de segundo hasta que se muestre (pista: `DELAY`)
2. Que exista una separación horizontal de 15 píxel entre el puntero del ratón y el *tooltip* (pista: `OFFSETX`)
3. Que el *tooltip* se muestre en la parte superior del puntero del ratón y no en la parte inferior (pista: `ABOVE`)

El nuevo aspecto del tooltip se muestra en la siguiente imagen:

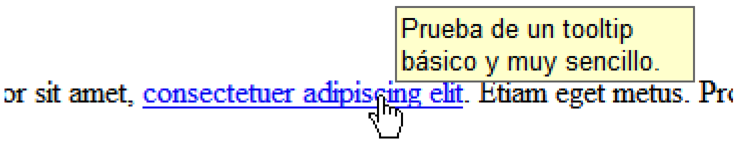


Figura 8.12. Nuevo aspecto del tooltip

8.4. Menú desplegable

La navegación de algunas páginas web se realiza mediante menús desplegables, que disponen de varios niveles jerárquicos que se despliegan a medida que el usuario pasa el puntero del ratón por encima de cada elemento. Aunque CSS permite realizar menús desplegables horizontales y verticales de cualquier nivel de profundidad, los navegadores de la familia Internet Explorer versión 6.0 y anteriores no disponen del suficiente soporte de CSS como para crear este tipo de menús.

De esta forma, la única forma de crear un menú desplegable que funcione correctamente en cualquier navegador consiste en utilizar JavaScript. A pesar de que realizar un menú desplegable sencillo con JavaScript no es una tarea muy compleja, se va a utilizar un componente ya realizado que pertenece a la librería de desarrollo web de Yahoo.

La *Yahoo UI Library*, conocida como YUI, y que se puede traducir como *"Librería de componentes de interfaz de usuario de Yahoo"* es un conjunto de utilidades y controles escritos en JavaScript para el desarrollo rápido y sencillo de aplicaciones web complejas.

La librería completa está dividida en módulos y componentes relacionados con CSS, DOM, eventos, AJAX, etc. Entre las utilidades disponibles se encuentran calendarios, *tooltips*, cuadros que autocompletan el texto, árboles jerárquicos, etc. Además de esas utilidades, la YUI incluye un completo módulo de menús que permite realizar decenas de tipos de menús diferentes: horizontales, verticales, desplegables, estáticos, menús contextuales, menús de aplicación, menús realizados con XHTML o con JavaScript, etc.

La librería YUI se puede descargar gratuitamente en http://developer.yahoo.com/yui/, la información específica sobre los menús se puede encontrar en http://developer.yahoo.com/yui/menu/ y se pueden ver decenas de ejemplos de menús ya realizados en http://developer.yahoo.com/yui/examples/menu/.

A continuación se indican los pasos necesarios para incluir un menú vertical desplegable:

1) Enlazar los archivos JavaScript y CSS requeridos:

La librería YUI se puede descargar gratuitamente con todos los archivos necesarios, toda la documentación y cientos de ejemplos (por lo que comprimida ya ocupa más de 11 MB). Aunque es posible utilizar los archivos CSS y JavaScript proporcionados por la librería desde nuestro servidor, Yahoo permite enlazar los archivos CSS y JavaScript desde sus propios servidores.

De esta forma, todos los archivos requeridos por las utilidades de Yahoo se pueden servir directamente desde sus servidores, mucho más rápidos y fiables que cualquier servidor particular. Como el usuario se descarga los archivos desde los servidores de Yahoo, el ahorro de ancho de banda es total y la aplicación carga mucho más rápido.

Para crear un menú vertical, se deben enlazar los siguientes archivos:

```
<!-- Dependencias -->
<link rel="stylesheet" type="text/css" href="http://yui.yahooapis.com/2.3.0/
build/menu/assets/skins/sam/menu.css">
<script type="text/javascript" src="http://yui.yahooapis.com/2.3.0/build/
yahoo-dom-event/yahoo-dom-event.js"></script>
<script type="text/javascript" src="http://yui.yahooapis.com/2.3.0/build/
container/container_core-min.js"></script>

<!-- Código para menús -->
<script type="text/javascript" src="http://yui.yahooapis.com/2.3.0/build/menu/
menu-min.js"></script>
```

Si se opta por servir los archivos desde nuestro propio servidor, es recomendable enlazar las versiones comprimidas de cada archivo, que se pueden reconocer porque su nombre termina en `-min.js`.

2) Definir el código XHTML de los elementos del menú:

```
<div id="menu_vertical" class="yuimenu">
<div class="bd">
  <ul class="first-of-type">
  <li class="yuimenuitem"><a href="#">Lorem ipsum</a></li>
  <li class="yuimenuitem"><a href="#">Dolor sit amet</a></li>
  <li class="yuimenuitem">Consectetuer
    <div id="consectetuer" class="yuimenu">
    <div class="bd">
      <ul>
        <li class="yuimenuitem"><a href="#">Donec quis nunc</a></li>
        <li class="yuimenuitem"><a href="#">Fusce eleifend</a></li>
        <li class="yuimenuitem"><a href="#">Donec erat</a></li>
        <li class="yuimenuitem"><a href="#">Faucibus orci</a></li>
        <li class="yuimenuitem"><a href="#">Volutpat quam</a></li>
      </ul>
    </div>
    </div>
  </li>
  <li class="yuimenuitem"><a href="#">Adipiscing elit</a></li>
  </ul>
</div>
</div>
```

La librería YUI dispone de muchos tipos diferentes de menús. Además, los menús se pueden construir completamente con JavaScript (en la página XHTML no aparecen los elementos del menú) o a partir del código XHTML de los elementos del menú incluido en la página.

Este último método, definir el menú con XHTML y después aplicarle JavaScript, es la forma recomendada, ya que si el usuario no tiene JavaScript activado, el menú se visualiza correctamente. Por ese motivo, no se va a estudiar el método de construcción de un menú de YUI exclusivamente con JavaScript.

Los elementos que forman los menús creados con YUI se construyen mediante listas no ordenadas con las etiquetas `<ul>` y `<li>`. Un pequeño inconveniente de los menús definidos con la librería YUI es que cada uno de los menús (el principal y todos los desplegables) debe encerrarse con dos elementos de tipo `<div>`. En otras palabras, el esquema de cada menú (y submenú desplegable) es el siguiente:

```
<div id="cualquier_identificador" class="yuimenu">
  <div class="bd">
    <ul class="first-of-type">
      ...
    </ul>
  </div>
</div>
```

Los valores de los atributos `class` de los elementos `<div>` anteriores deben mantenerse para que el menú funcione correctamente.

3) Construir el menú y mostrarlo en la página:

```
<script type="text/javascript">
YAHOO.util.Event.onContentReady("menu_vertical", function () {
```

```
    var elMenu = new YAHOO.widget.Menu("menu_vertical", { width: '150px' });
    elMenu.render();
    elMenu.show();
  });
  </script>
```

El código JavaScript que crea dinámicamente el menú requiere que el árbol DOM de la página se haya construido completamente, por lo que se utiliza la función `onContentReady()` que forma parte de las utilidades para eventos de la librería YUI.

Para construir el menú se crea un nuevo objeto de tipo `YAHOO.widget.Menu` y se le pasan las opciones de inicialización (en este caso, que tenga una anchura de 150 píxel).:

```
var elMenu = new YAHOO.widget.Menu("menu_vertical", { width: '150px' });
```

Una vez creado el objeto del menú, se construye mediante la instrucción `elMenu.render()` y se muestra en la página con la instrucción `elMenu.show()`.

El resultado final es un menú vertical desplegable con el siguiente aspecto:

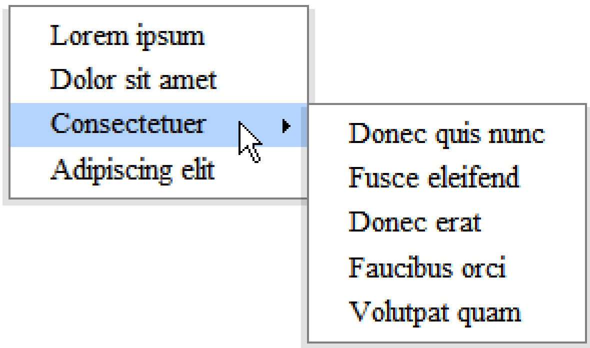


Figura 8.13. Aspecto final del menú vertical desplegable creado con la librería Yahoo YUI

8.5. Galerías de imágenes (Lightbox)

Muchos sitios web utilizan galerías de imágenes para mostrar sus productos y servicios. Este tipo de galerías muestran una serie de miniaturas de imágenes que se amplían al pinchar sobre cada imagen. Hasta hace poco, la técnica más utilizada para construir una galería consistía en incluir las miniaturas en la página y abrir cada imagen grande en una ventana emergente (o *pop-up*) o en una nueva página.

El uso de técnicas basadas en JavaScript ha supuesto una revolución en la creación de las galerías de imágenes y ha permitido mejorar la experiencia de navegación del usuario. La técnica se conoce como *Lightbox* y fue creada originalmente por Lokesh Dhakar. Lightbox es una técnica muy sencilla de utilizar, funciona correctamente en todos los navegadores y permite mantener la semántica del documento (no *ensucia* la página con código JavaScript).

El código de la versión más reciente se puede descargar gratuitamente en http://www.huddletogether.com/projects/lightbox2/, donde también se puede ver una demostración de las galerías de imágenes construidas con Lightbox.

La descarga incluye el código fuente del script, todos los archivos JavaScript externos necesarios, archivos CSS e imágenes de prueba y una breve pero completa documentación.

A continuación se indican los pasos necesarios para incluir Lightbox en una página web:

1) Enlazar los archivos JavaScript y CSS requeridos:

```
<!-- Dependencias -->
<link rel="stylesheet" href="css/lightbox.css" type="text/css" media="screen" />
<script type="text/javascript" src="js/prototype.js"></script>
<script type="text/javascript" src="js/
scriptaculous.js?load=effects,builder"></script>

<!-- Código fuente -->
<script type="text/javascript" src="js/lightbox.js"></script>
```

Lightbox utiliza dos de las librerías más conocidas y utilizadas para realizar aplicaciones JavaScript complejas: Prototype y Scriptaculous. Además de estos dos archivos JavaScript, también es necesario enlazar un archivo de tipo CSS que controla el aspecto de Lightbox y enlaza las imágenes utilizadas.

2) Activar Lightbox:

La página XHTML original contiene un enlace simple que abre una nueva página en la que se puede ver ampliada la imagen original:

```
<a href="images/image-1.jpg" title="Título de la imagen 1">Imagen 1</a>
```

Para activar Lightbox, tan solo es necesario incluir el siguiente atributo `rel`:

```
<a href="images/image-1.jpg" rel="lightbox" title="Título de la imagen
1">Imagen 1</a>
```

Ahora, si el usuario tiene activado JavaScript, al pinchar en el enlace:



Figura 8.14. Enlace con el atributo "lightbox" añadido

Se mostrará la imagen ampliada centrada en la misma página:



Figura 8.15. Imagen ampliada que se muestra al pinchar sobre el enlace

Lightbox se activa automáticamente para todos los enlaces cuyo atributo `rel` sea igual a `lightbox`. Además, Lightbox muestra como título de la imagen el valor del atributo `title` del enlace. Si el usuario no tiene activado JavaScript, la imagen se abrirá en una página nueva, como ocurre si no se utiliza Lightbox. La sencillez de uso y el resultado tan espectacular que se consigue son los puntos fuertes de Lightbox.

Normalmente, Lightbox no se utiliza con enlaces de texto sino con imágenes que contienen enlaces:

```
<a href="images/image-1.jpg" rel="lightbox" title="Título de la imagen 1"><img
src="images/imagen_pequena1.jpg" /></a>
```

Ahora, si el usuario tiene activado JavaScript, al pinchar sobre la imagen pequeña:

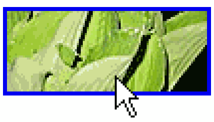


Figura 8.16. Imagen en miniatura con el enlace preparado para Lightbox

Se mostrará la imagen ampliada centrada en la misma página:

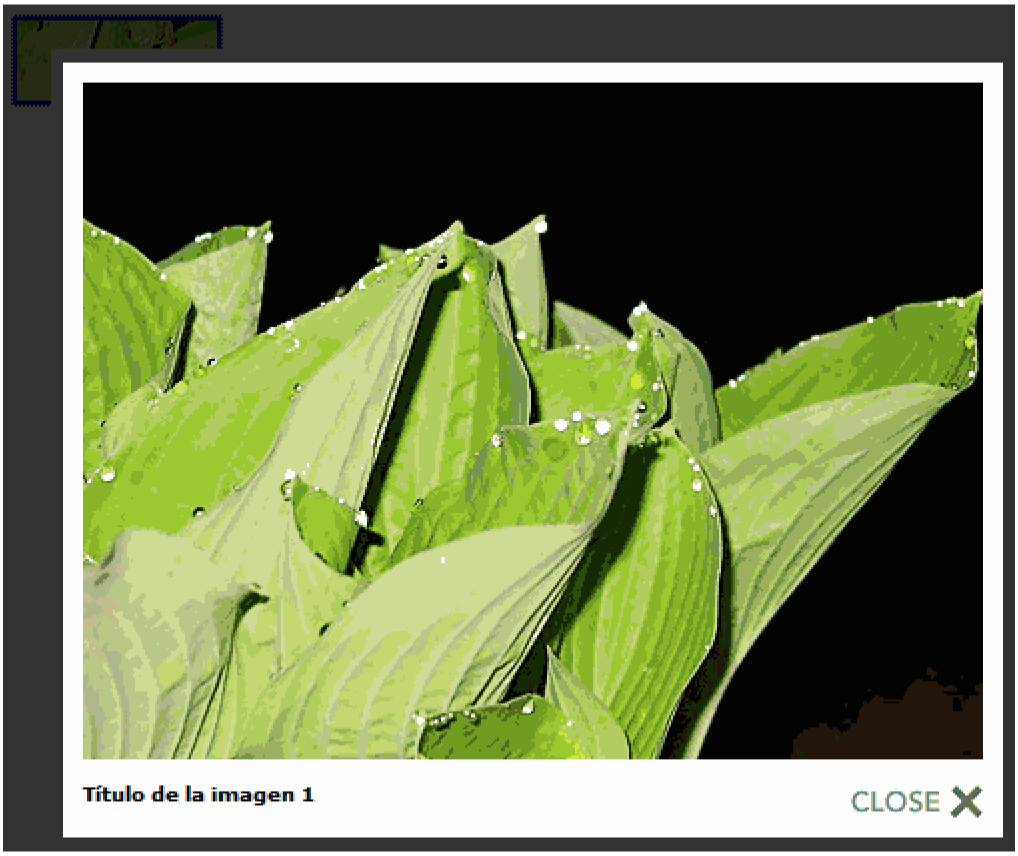


Figura 8.17. Imagen ampliada que se muestra al pinchar sobre la imagen en miniatura

La imagen ampliada se muestra con una vistosa animación y el fondo de la página completa se oscurece para que el usuario centre toda su atención en la imagen. El usuario puede cerrar la imagen ampliada y volver a la página original pulsando sobre la imagen `CLOSE X` de la esquina inferior derecha de la imagen.

Utilizar Lightbox aporta muchas ventajas respecto de la experiencia de usuario del sitio web. Además, el código XHTML permanece intacto y no se ensucia con llamadas a funciones JavaScript, ya que lo único que hay que añadir es un atributo `rel="lightbox"` a las imágenes y enlaces que vayan a utilizar Lightbox.

Además, Lightbox permite relacionar varias imágenes entre sí para crear una galería de imágenes. Así, mientras se visualiza una imagen ampliada es posible pasar a la siguiente imagen de la galería. Para relacionar varias imágenes entre sí, tan sólo es necesario aplicar un valor único al atributo `rel` como se muestra a continuación:

```
<a href="images/imagen-1.jpg" rel="lightbox[galeria1]">imagen #1</a>
<a href="images/imagen-2.jpg" rel="lightbox[galeria1]">imagen #2</a>
<a href="images/imagen-3.jpg" rel="lightbox[galeria1]">imagen #3</a>
```

Todas las imágenes que tengan el mismo valor del atributo `rel`, automáticamente pasarán a formar parte de una galería de fotos por la que se puede navegar hacia la siguiente o hacia la anterior imagen.

La técnica Lightbox presentada solamente es válida para elementos de tipo imagen. No es posible por tanto mostrar una página web o cierto código XHTML dentro de una ventana de ese tipo.

Afortunadamente, existen otras librerías basadas en Lightbox y que permiten mostrar cualquier tipo de código XHMTL en el interior de la ventana. La más conocida es la técnica creada por ParticleTree y que se puede encontrar en: http://particletree.com/features/lightbox-gone-wild/

Capítulo 9. Detección y corrección de errores

JavaScript es un lenguaje de programación interpretado, lo que significa que no se pueden detectar la mayoría de errores en el código hasta que se ejecutan los scripts. De esta forma, antes de considerar un script como correcto, es necesario probarlo en todos los navegadores sobre los que se vaya a utilizar.

Cuando se producen errores durante la ejecución de un script, los navegadores proporcionan cierta información útil para descubrir el punto exacto en el que se ha producido el error y su posible solución. Solucionar los errores de un script se denomina *"depurar el script"* o *"debugear el script"* (término que viene de la palabra inglesa *"debug"*, que significa "eliminar los errores de una aplicación").

Desafortunadamente, no todos los navegadores proporcionan la misma información útil, lo que complica la solución de los errores para cada tipo de navegador. A continuación se muestran las herramientas que proporciona cada navegador para detectar y corregir los errores en JavaScript.

9.1. Corrección de errores con Internet Explorer

Dependiendo de su configuración, el navegador Internet Explorer puede tener desactivada la notificación de errores de JavaScript. Por este motivo, en primer lugar puede ser necesario activar los mensajes de aviso sobre los errores de JavaScript. Para activar las notificaciones, se accede al menú `Herramientas > Opciones`, pestaña `Opciones Avanzadas` y se activa la opción `Mostrar una notificación sobre cada error de secuencia de comandos`, como se muestra en la siguiente imagen:

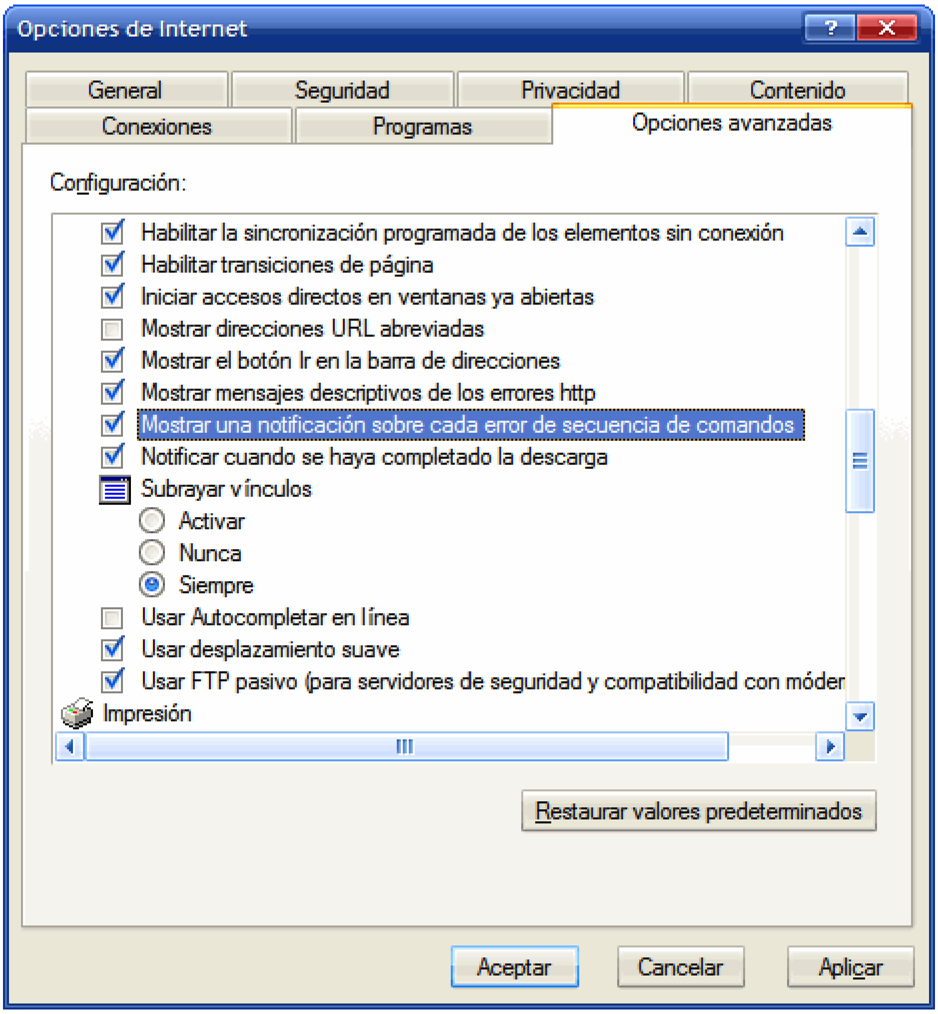


Figura 9.1. Opciones avanzadas en Internet Explorer necesarias para activar la notificación de errores de JavaScript

Una vez activado, se mostrará un mensaje de error cada vez que se produzcan errores de JavaScript en una página. Además, en la esquina inferior izquierda se muestra un pequeño mensaje indicando que la página contiene errores:

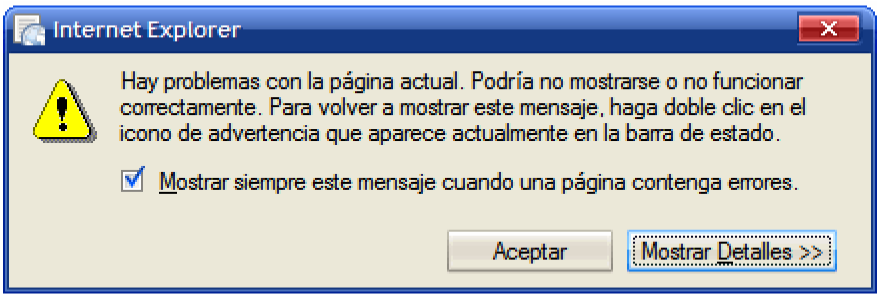


Figura 9.2. Mensaje de aviso de Internet Explorer sobre los errores de JavaScript

Pulsando sobre el botón `Mostrar Detalles >>` es posible acceder a cierta información adicional sobre el error producido:

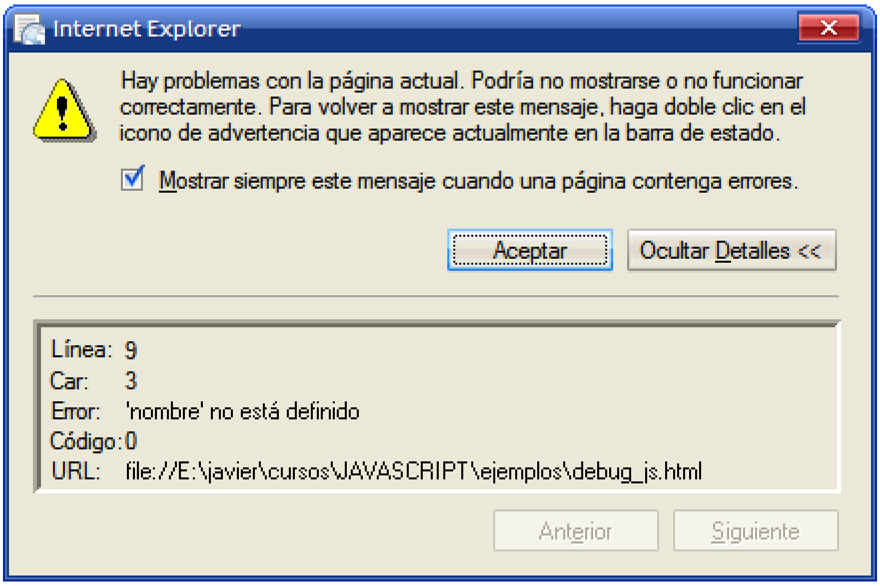


Figura 9.3. Información adicional proporcionada por Internet Explorer sobre los errores de JavaScript

Internet Explorer es el navegador que menos información proporciona. Además, cuando se produce un error no siempre indica correctamente la posición del posible error.

El tipo de mensajes (muy breves y en ocasiones ambiguos) y la falta de precisión sobre el lugar en el que se ha producido realmente el error, hacen que depurar un script en Internet Explorer sea una tarea excesivamente complicada.

9.2. Corrección de errores con Firefox

Depurar scripts utilizando Firefox es una experiencia completamente diferente y más sencilla que depurarlos con Internet Explorer. Firefox proporciona herramientas más útiles, activadas por defecto y que muestran más información y mucho más precisa.

Para depurar un script con Firefox, solamente es necesario acceder a la opción `Consola de error` dentro del menú `Herramientas`:

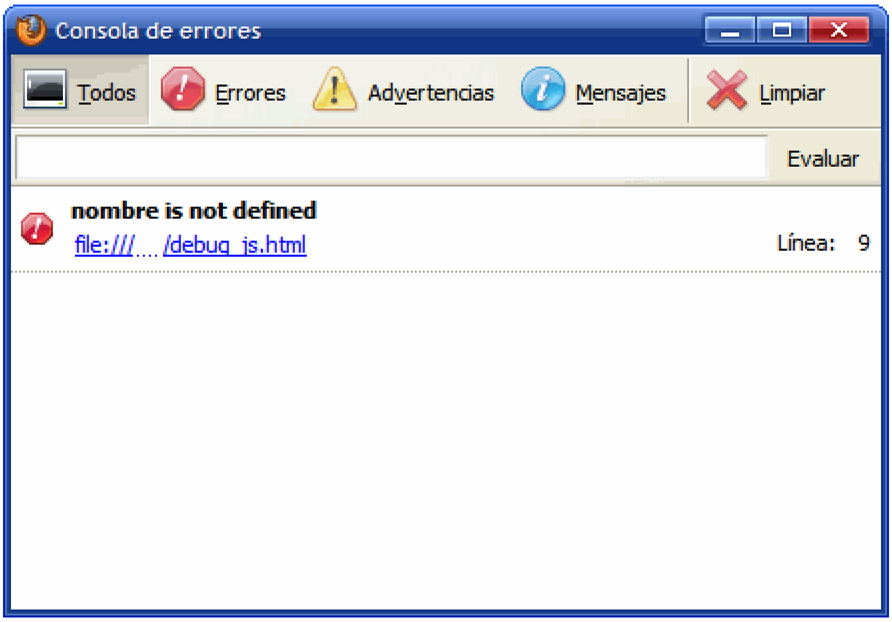


Figura 9.4. Consola de errores de Firefox donde se muestra toda la información sobre los errores de JavaScript

La consola de errores permite diferenciar los mensajes de información, los mensajes de aviso y los mensajes de error. Además, permite visualizar todos los errores de la página simultáneamente. Por cada error detectado se indica la posible solución mediante un mensaje en inglés y se muestra el trozo de código del script donde se ha producido el error. Además, pulsando sobre el enlace incluido se accede a la línea concreta del archivo concreto donde se ha producido el error.

Además, Firefox permite instalar pequeñas mejoras y ampliaciones en el navegador, que se conocen con el nombre de extensiones. Una de las extensiones más interesantes para los desarrolladores de aplicaciones web es Firebug, que se puede descargar gratuitamente desde http://www.getfirebug.com/ El mismo error de JavaScript muestra la siguiente información en Firebug:

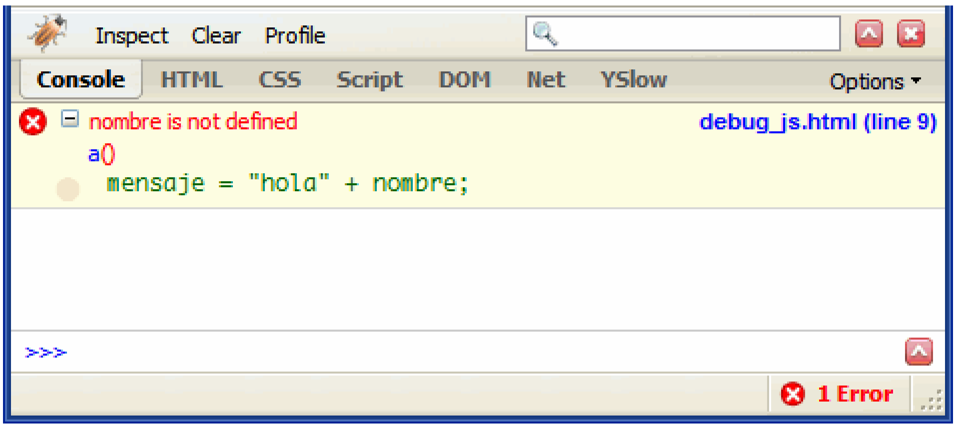


Figura 9.5. Información mostrada por Firebug sobre un error de JavaScript

Si se pincha sobre el mensaje de error, Firebug también redirige a la línea concreta del script en la que se está produciendo el error:

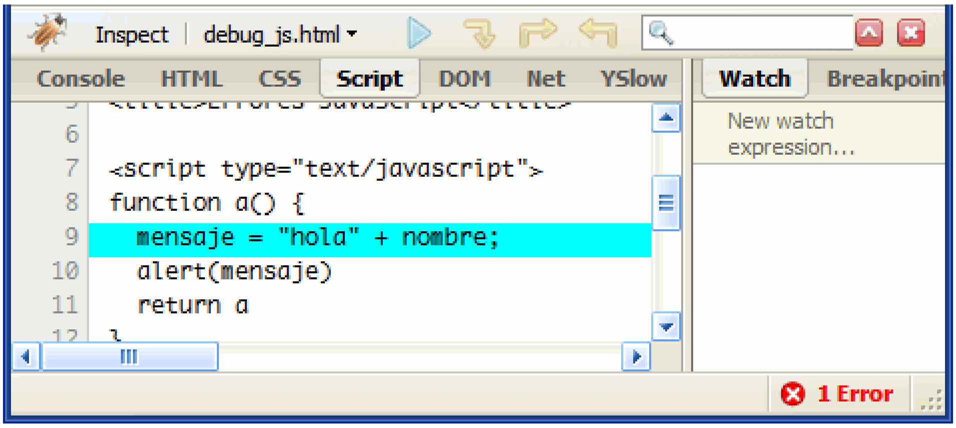


Figura 9.6. Firebug permite ir directamente a la línea concreta del script en la que se está produciendo el error

Firebug incluye cientos de utilidades y herramientas necesarias para depurar aplicaciones web y para diseñar páginas web. Además, proporciona información detallada sobre XHTML, CSS, DOM y JavaScript. Toda la documentación, tutoriales y preguntas frecuentes sobre Firebug se pueden encontrar en http://www.getfirebug.com/docs.html

9.3. Corrección de errores con Opera

El navegador Opera también dispone de una consola de errores muy completa que muestra toda la información sobre el error producido. La consola de errores se accede desde el menú `Herramientas > Avanzado` y la opción `Consola de error`:

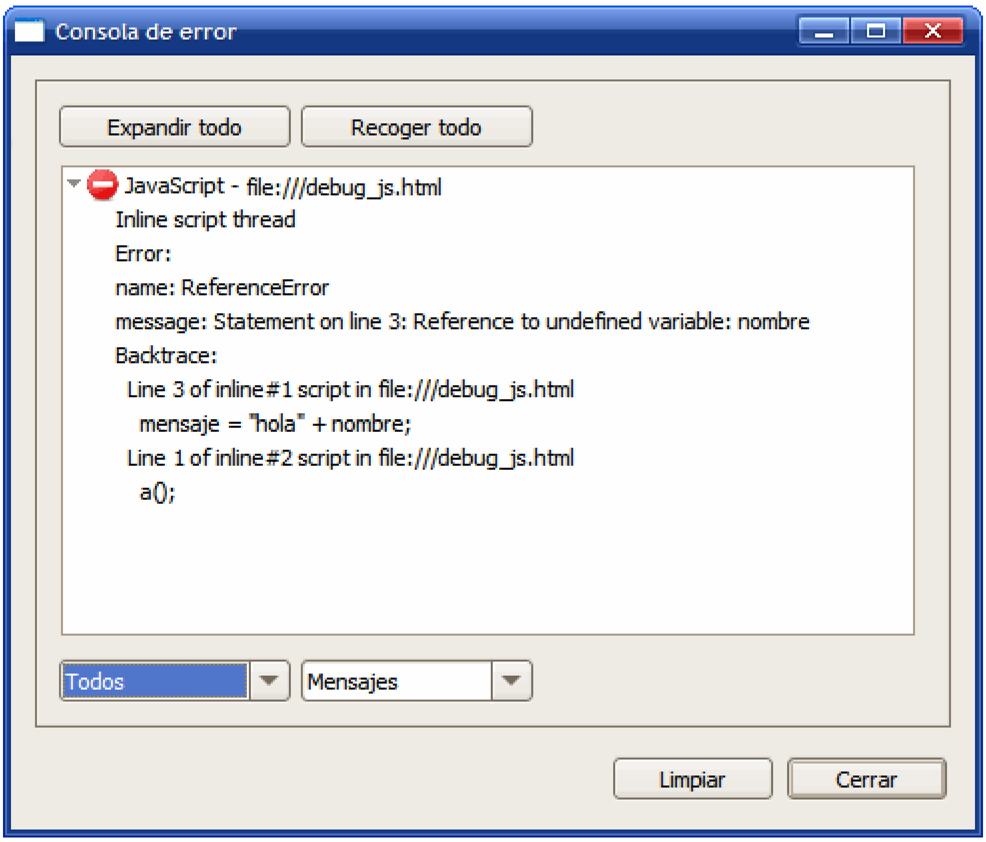


Figura 9.7. Consola de errores de Opera donde se muestra toda la información sobre los errores de JavaScript

Capítulo 10. Recursos útiles

Estándares y especificaciones oficiales

- Especificación oficial del lenguaje JavaScript: ECMAScript Language Specification (http://www.ecma-international.org/publications/standards/Ecma-262.htm) (3ª edición, Diciembre de 1999)
- Otros estándares relacionados: ECMAScript Components Specification (ECMA-290) (http://www.ecma-international.org/publications/standards/Ecma-290.htm) , ECMAScript 3rd Edition Compact Profile (ECMA-327) (http://www.ecma-international.org/publications/standards/Ecma-327.htm) , ECMAScript for XML (E4X) Specification (ECMA-357) (http://www.ecma-international.org/publications/standards/Ecma-357.htm)

Scripts y utilidades gratuitas

- Hotscripts (http://www.hotscripts.com/JavaScript/index.html) : sitio web en inglés que contiene referencias a miles de scripts y utilidades de JavaScript gratuitas.
- Dynamic Drive (http://www.dynamicdrive.com/) : sitio web en inglés con cientos de scripts gratuitos de JavaScript con las utilidades más comunes (calendarios, menús, formularios, animaciones de texto, enlaces, galerías de imágenes, etc.)
- Gamarod JavaScript (http://www.gamarod.com.ar/) : sitio web en español que contiene decenas de utilidades JavaScript gratuitas listas para usar.

Libros

- Professional JavaScript for Web Developers (http://www.wrox.com/WileyCDA/WroxTitle/productCd-0764579088.html) , Nicholas C. Zakas (ISBN: 978-0-7645-7908-0). Algunos capítulos son demasiado avanzados para los principiantes de JavaScript, pero la mayoría de capítulos explican conceptos básicos de forma sencilla y clara.
- JavaScript: The Definitive Guide (Fifth Edition) (http://www.oreilly.com/catalog/jscript5/index.html) , David Flanagan (ISBN 13: 9780596101992). Referencia completa de JavaScript en más de 1.000 páginas que explican con detalle cualquier aspecto de este lenguaje de programación.

Capítulo 11. Ejercicios resueltos

11.1. Ejercicio 1

Página HTML:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Ejercicio 1 - Archivo externo y varios mensajes</title>
<script type="text/javascript" src="codigo.js"></script>
</head>

<body>
<noscript>Esta página <strong>requiere</strong> el uso de JavaScript y parece
que está desactivado</noscript>
<p>Esta página muestra 2 mensajes</p>
</body>
</html>
```

Archivo `codigo.js`

```
// Al cargarse el archivo JavaScript, se muestra un mensaje
alert("Hola Mundo!");

// Despues del primer mensaje, se muestra otro mensaje seguido
alert("Soy el primer script");
```

11.2. Ejercicio 2

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Ejercicio 2 - Mostrar mensajes complejos</title>

<script type="text/javascript">
  var mensaje = "Hola Mundo! \n Qué facil es incluir \'comillas simples\' \n y
\"comillas dobles\" ";
  alert(mensaje);
</script>
</head>

<body>
<p>Esta página muestra un mensaje complejo</p>
</body>
</html>
```

11.3. Ejercicio 3

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Ejercicio 3 - Arrays simples</title>
<script type="text/javascript">
// Array que almacena los 12 meses del año
var meses = ["Enero", "Febrero", "Marzo", "Abril", "Mayo", "Junio", "Julio",
"Agosto", "Septiembre", "Octubre", "Noviembre", "Diciembre"];

// Se muestra el nombre de cada mes
alert(meses[0]);
alert(meses[1]);
alert(meses[2]);
alert(meses[3]);
alert(meses[4]);
alert(meses[5]);
alert(meses[6]);
alert(meses[7]);
alert(meses[8]);
alert(meses[9]);
alert(meses[10]);
alert(meses[11]);
</script>
</head>

<body>
<p>Esta página muestra los 12 meses del año</p>
</body>
</html>
```

11.4. Ejercicio 4

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Ejercicio 4 - Operadores</title>
<script type="text/javascript">
var valores = [true, 5, false, "hola", "adios", 2];

// Cual de los 2 elementos de texto es mayor
// Si el resultado es true, el primer texto es mayor
var resultado = valores[3] > valores[4];
alert(resultado);

// Combinar valores booleanos
var valor1 = valores[0];
var valor2 = valores[2];

// Obtener un resultado TRUE
```

```
var resultado = valor1 || valor2;
alert(resultado);

// Obtener un resultado FALSE
resultado = valor1 && valor2;
alert(resultado);

// Operaciones matemáticas
var num1 = valores[1];
var num2 = valores[5];

var suma = num1 + num2;
alert(suma);

var resta = num1 - num2;
alert(resta);

var multiplicacion = num1 * num2;
alert(multiplicacion);

var division = num1 / num2;
alert(division);

var modulo = num1 % num2;
alert(modulo);

</script>
</head>

<body>
<p>Esta página muestra el uso básico de los operadores</p>
</body>
</html>
```

11.5. Ejercicio 5

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Ejercicio 5 - Estructuras de control, IF</title>
<script type="text/javascript">
var numero1 = 5;
var numero2 = 8;

if(numero1 <= numero2) {
  alert("numero1 no es mayor que numero2");
}
if(numero2 >= 0) {
  alert("numero2 es positivo");
}
if(numero1 < 0 || numero1 != 0) {
  alert("numero1 es negativo o distinto de cero");
}
```

```
if(++numero1 < numero2) {
  alert("Incrementar en 1 unidad el valor de numero1 no lo hace mayor o igual
que numero2");
}
</script>
</head>

<body>
<p>Esta página muestra el uso básico de la estructura de control IF</p>
</body>
</html>
```

11.6. Ejercicio 6

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Ejercicio 6 - Calculo de la letra del DNI</title>
<script type="text/javascript">
var letras = ['T', 'R', 'W', 'A', 'G', 'M', 'Y', 'F', 'P', 'D', 'X', 'B', 'N',
'J', 'Z', 'S', 'Q', 'V', 'H', 'L', 'C', 'K', 'E', 'T'];

var numero = prompt("Introduce tu número de DNI (sin la letra)");
var letra = prompt("Introduce la letra de tu DNI (en mayúsculas)");
letra = letra.toUpperCase();

if(numero < 0 || numero > 99999999) {
  alert("El número proporcionado no es válido");
}
else {
  var letraCalculada = letras[numero % 23];
  if(letraCalculada != letra) {
    alert("La letra o el número proporcionados no son correctos");
  }
  else {
    alert("El número de DNI y su letra son correctos");
  }
}

</script>
</head>

<body>
<p>Esta página calcula la letra del DNI</p>
</body>
</html>
```

11.7. Ejercicio 7

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
```

```
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Ejercicio 7 - Factorial de un número entero</title>

<script type="text/javascript">
var numero = prompt("Introduce un número y se mostrará su factorial");
var resultado = 1;

for(var i=1; i<=numero; i++) {
  resultado *= i;
}
alert(resultado);
</script>
</head>

<body>
</body>
</html>
```

11.8. Ejercicio 8

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Ejercicio 8 - Funciones básicas</title>

<script type="text/javascript">
var numero = prompt("Introduce un número entero");

var resultado = parImpar(numero);
alert("El número "+numero+" es "+resultado);

function parImpar(numero) {
  if(numero % 2 == 0) {
    return "par";
  }
  else {
    return "impar";
  }
}
</script>
</head>

<body>
</body>
</html>
```

11.9. Ejercicio 9

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
```

```
<title>Ejercicio 9 - Información sobre cadenas</title>

<script type="text/javascript">
function info(cadena) {

  var resultado = "La cadena \""+cadena+"\" ";

  // Comprobar mayúsculas y minúsculas
  if(cadena == cadena.toUpperCase()) {
    resultado += " está formada sólo por mayúsculas";
  }
  else if(cadena == cadena.toLowerCase()) {
    resultado += " está formada sólo por minúsculas";
  }
  else {
    resultado += " está formada por mayúsculas y minúsculas";
  }

  return resultado;
}

alert(info("OVNI = OBJETO VOLADOR NO IDENTIFICADO"));
alert(info("En un lugar de la mancha..."));
</script>
</head>

<body>
</body>
</html>
```

11.10. Ejercicio 10

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Ejercicio 10 - Detección de palíndromos</title>

<script type="text/javascript">
function palindromo(cadena) {

  var resultado = "La cadena \""+cadena+"\" \n";

  // Pasar a minusculas la cadena
  var cadenaOriginal = cadena.toLowerCase();

  // Convertir la cadena en un array
  var letrasEspacios = cadenaOriginal.split("");

  // Eliminar los espacios en blanco (este paso es demasiado largo ya que no se
utiliza la funcion "replace")
  var cadenaSinEspacios = "";
  for(i in letrasEspacios) {
    if(letrasEspacios[i] != " ") {
```

```
      cadenaSinEspacios += letrasEspacios[i];
    }
  }

  var letras = cadenaSinEspacios.split("");
  var letrasReves = cadenaSinEspacios.split("").reverse();

  // Este paso tambien es muy largo porque no se utiliza la sentencia "break"
  var iguales = true;
  for(i in letras) {
    if(letras[i] == letrasReves[i]) {
      // Todo bien
    }
    else {
      // Alguna letra es distinta, por lo que ya no es un palindromo
      iguales = false;
    }
  }

  if(iguales) {
    resultado += " es un palíndromo";
  }
  else {
    resultado += " no es un palíndromo";
  }

  return resultado;
}

alert(palindromo("La ruta nos aporto otro paso natural"));
alert(palindromo("Esta frase no se parece a ningun palindromo"));
</script>
</head>

<body>
</body>
</html>
```

11.11. Ejercicio 11

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Ejercicio 11 - DOM básico</title>

<script type="text/javascript">
window.onload = function() {
  var info = document.getElementById("informacion");

  // Numero de enlaces de la pagina
  var enlaces = document.getElementsByTagName("a");
  info.innerHTML = "Numero de enlaces = "+enlaces.length;
```

```
  // Direccion del penultimo enlace
  var mensaje = "El penultimo enlace apunta a: "+enlaces[enlaces.length-2].href;
  info.innerHTML = info.innerHTML + "<br/>" + mensaje;

  // Numero de enlaces que apuntan a http://prueba
  var contador = 0;
  for(var i=0; i<enlaces.length; i++) {
    // Es necesario comprobar los enlaces http://prueba y
    // http://prueba/ por las diferencias entre navegadores
    if(enlaces[i].href == "http://prueba" || enlaces[i].href ==
"http://prueba/") {
      contador++;
    }
  }
  info.innerHTML = info.innerHTML + "<br/>" + contador + " enlaces apuntan a
http://prueba"

  // Numero de enlaces del tercer párrafo
  var parrafos = document.getElementsByTagName("p");
  enlaces = parrafos[2].getElementsByTagName("a");
  info.innerHTML = info.innerHTML + "<br/>" + "Numero de enlaces del tercer
párrafo = "+enlaces.length;
}
</script>
</head>

<body>
<div id="informacion" style="border:thin solid silver; padding:.5em"></div>
<p>Lorem ipsum dolor sit amet, <a href="http://prueba">consectetuer adipiscing
elit</a>. Sed mattis enim vitae orci. Phasellus libero. Maecenas nisl arcu,
consequat congue, commodo nec, commodo ultricies, turpis. Quisque sapien nunc,
posuere vitae, rutrum et, luctus at, pede. Pellentesque massa ante, ornare id,
aliquam vitae, ultrices porttitor, pede. Nullam sit amet nisl elementum elit
convallis malesuada. Phasellus magna sem, semper quis, faucibus ut, rhoncus
non, mi. <a href="http://prueba2">Fusce porta</a>. Duis pellentesque, felis eu
adipiscing ullamcorper, odio urna consequat arcu, at posuere ante quam non
dolor. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis
scelerisque. Donec lacus neque, vehicula in, eleifend vitae, venenatis ac,
felis. Donec arcu. Nam sed tortor nec ipsum aliquam ullamcorper. Duis accumsan
metus eu urna. Aenean vitae enim. Integer lacus. Vestibulum venenatis erat eu
odio. Praesent id metus.</p>

<p>Aenean at nisl. Maecenas egestas dapibus odio. Vestibulum ante ipsum primis
in faucibus orci luctus et ultrices posuere cubilia Curae; Proin consequat
auctor diam. <a href="http://prueba">Ut bibendum blandit est</a>. Curabitur
vestibulum. Nunc malesuada porttitor sapien. Aenean a lacus et metus venenatis
porta. Suspendisse cursus, sem non dapibus tincidunt, lorem magna porttitor
felis, id sodales dolor dolor sed urna. Sed rutrum nulla vitae tellus. Sed quis
eros nec lectus tempor lacinia. Aliquam nec lectus nec neque aliquet dictum.
Etiam <a href="http://prueba3">consequat sem quis massa</a>. Donec aliquam
euismod diam. In magna massa, mattis id, pellentesque sit amet, porta sit amet,
lectus. Curabitur posuere. Aliquam in elit. Fusce condimentum, arcu in
scelerisque lobortis, ante arcu scelerisque mi, at cursus mi risus sed
tellus.</p>
```

```
<p>Donec sagittis, nibh nec ullamcorper tristique, pede velit feugiat massa, at
sollicitudin justo tellus vitae justo. Vestibulum aliquet, nulla sit amet
imperdiet suscipit, nunc erat laoreet est, a <a href="http://prueba">aliquam
leo odio sed sem</a>. Quisque eget eros vehicula diam euismod tristique. Ut
dui. Donec in metus sed risus laoreet sollicitudin. Proin et nisi non arcu
sodales hendrerit. In sem. Cras id augue eu lorem dictum interdum. Donec
pretium. Proin <a href="http://prueba4">egestas</a> adipiscing ligula. Duis
iaculis laoreet turpis. Mauris mollis est sit amet diam. Curabitur hendrerit,
eros quis malesuada tristique, ipsum odio euismod tortor, a vestibulum nisl mi
at odio. <a href="http://prueba5">Sed non lectus non est pellentesque</a>
auctor.</p>
</body>
</html>
```

11.12. Ejercicio 12

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Ejercicio 12 - DOM básico y atributos XHTML</title>

<style type="text/css">
.oculto { display: none; }
.visible { display: inline; }
</style>

<script type="text/javascript">
function muestra() {
  var elemento = document.getElementById("adicional");
  elemento.className = "visible";

  var enlace = document.getElementById("enlace");
  enlace.className = "oculto";
}
</script>
</head>

<body>

<p id="texto">Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Sed
mattis enim vitae orci. Phasellus libero. Maecenas nisl arcu, consequat congue,
commodo nec, commodo ultricies, turpis. Quisque sapien nunc, posuere vitae,
rutrum et, luctus at, pede. Pellentesque massa ante, ornare id, aliquam vitae,
ultrices porttitor, pede. <span id="adicional" class="oculto">Nullam sit amet
nisl elementum elit convallis malesuada. Phasellus magna sem, semper quis,
faucibus ut, rhoncus non, mi. Duis pellentesque, felis eu adipiscing
ullamcorper, odio urna consequat arcu, at posuere ante quam non dolor. Lorem
ipsum dolor sit amet, consectetuer adipiscing elit. Duis scelerisque. Donec
lacus neque, vehicula in, eleifend vitae, venenatis ac, felis. Donec arcu. Nam
sed tortor nec ipsum aliquam ullamcorper. Duis accumsan metus eu urna. Aenean
vitae enim. Integer lacus. Vestibulum venenatis erat eu odio. Praesent id
metus.</span></p>
<a id="enlace" href="#" onclick="muestra(); return false;">Seguir leyendo</a>
```

```
</body>
</html>
```

11.13. Ejercicio 13

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Ejercicio 13 - DOM básico y atributos XHTML</title>

<script type="text/javascript">
function anade() {
  var elemento = document.createElement("li");
  var texto = document.createTextNode("Elemento de prueba");
  elemento.appendChild(texto);

  var lista = document.getElementById("lista");
  lista.appendChild(elemento);

  var nuevoElemento = "<li>Texto de prueba</li>";
  lista.innerHTML = lista.innerHTML + nuevoElemento;
}
</script>
</head>

<body>

<ul id="lista">
  <li>Lorem ipsum dolor sit amet</li>
  <li>Consectetuer adipiscing elit</li>
  <li>Sed mattis enim vitae orci</li>
  <li>Phasellus libero</li>
  <li>Maecenas nisl arcu</li>
</ul>

<input type="button" value="Añadir elemento" onclick="anade();">

</body>
</html>
```

11.14. Ejercicio 14

Si se utilizan manejadores de eventos como atributos de XHTML, la solución podría ser la siguiente:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Ejercicio 14 - DOM básico y atributos XHTML</title>
```

```
<script type="text/javascript">
function muestraOculta(id) {
  var elemento = document.getElementById('contenidos_'+id);
  var enlace = document.getElementById('enlace_'+id);

  if(elemento.style.display == "" || elemento.style.display == "block") {
    elemento.style.display = "none";
    enlace.innerHTML = 'Mostrar contenidos';
  }
  else {
    elemento.style.display = "block";
    enlace.innerHTML = 'Ocultar contenidos';
  }
}
</script>
</head>

<body>

<p id="contenidos_1">[1] Lorem ipsum dolor sit amet, consectetuer adipiscing
elit. Sed mattis enim vitae orci. Phasellus libero. Maecenas nisl arcu,
consequat congue, commodo nec, commodo ultricies, turpis. Quisque sapien nunc,
posuere vitae, rutrum et, luctus at, pede. Pellentesque massa ante, ornare id,
aliquam vitae, ultrices porttitor, pede. Nullam sit amet nisl elementum elit
convallis malesuada. Phasellus magna sem, semper quis, faucibus ut, rhoncus
non, mi. Duis pellentesque, felis eu adipiscing ullamcorper, odio urna
consequat arcu, at posuere ante quam non dolor. Lorem ipsum dolor sit amet,
consectetuer adipiscing elit. Duis scelerisque.</p>
<a id="enlace_1" href="#" onclick="muestraOculta('1'); return false;">Ocultar
contenidos</a>

<br/>

<p id="contenidos_2">[2] Lorem ipsum dolor sit amet, consectetuer adipiscing
elit. Sed mattis enim vitae orci. Phasellus libero. Maecenas nisl arcu,
consequat congue, commodo nec, commodo ultricies, turpis. Quisque sapien nunc,
posuere vitae, rutrum et, luctus at, pede. Pellentesque massa ante, ornare id,
aliquam vitae, ultrices porttitor, pede. Nullam sit amet nisl elementum elit
convallis malesuada. Phasellus magna sem, semper quis, faucibus ut, rhoncus
non, mi. Duis pellentesque, felis eu adipiscing ullamcorper, odio urna
consequat arcu, at posuere ante quam non dolor. Lorem ipsum dolor sit amet,
consectetuer adipiscing elit. Duis scelerisque.</p>
<a id="enlace_2" href="#" onclick="muestraOculta('2'); return false;">Ocultar
contenidos</a>

<br/>
<p id="contenidos_3">[3] Lorem ipsum dolor sit amet, consectetuer adipiscing
elit. Sed mattis enim vitae orci. Phasellus libero. Maecenas nisl arcu,
consequat congue, commodo nec, commodo ultricies, turpis. Quisque sapien nunc,
posuere vitae, rutrum et, luctus at, pede. Pellentesque massa ante, ornare id,
aliquam vitae, ultrices porttitor, pede. Nullam sit amet nisl elementum elit
convallis malesuada. Phasellus magna sem, semper quis, faucibus ut, rhoncus
non, mi. Duis pellentesque, felis eu adipiscing ullamcorper, odio urna
consequat arcu, at posuere ante quam non dolor. Lorem ipsum dolor sit amet,
```

```
consectetuer adipiscing elit. Duis scelerisque.</p>
<a id="enlace_3" href="#" onclick="muestraOculta('3'); return false;">Ocultar
contenidos</a>

</body>
</html>
```

También es posible solucionar el ejercicio utilizando manejadores de eventos *semánticos*:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Ejercicio 14 - DOM básico y atributos XHTML</title>

<script type="text/javascript">
window.onload = function() {
  var enlaces = document.getElementsByTagName('a');
  for(i in enlaces) {
    enlaces[i].onclick = muestraOculta;
  }
}

function muestraOculta() {
  var idEnlace = this.id;
  var trozos = idEnlace.split('_');
  var numero = trozos[1];
  var parrafo = document.getElementById('contenidos_' + numero);

  switch(parrafo.style.display) {
    case 'none':
      parrafo.style.display = 'block';
      this.innerHTML = 'Ocultar contenidos';
      break;
    case 'block':
    case '':
      parrafo.style.display = 'none';
      this.innerHTML = 'Mostrar contenidos';
      break;
  }
}
</script>
</head>

<body>

<p id="contenidos_1">[1] Lorem ipsum dolor sit amet, consectetuer adipiscing
elit. Sed mattis enim vitae orci. Phasellus libero. Maecenas nisl arcu,
consequat congue, commodo nec, commodo ultricies, turpis. Quisque sapien nunc,
posuere vitae, rutrum et, luctus at, pede. Pellentesque massa ante, ornare id,
aliquam vitae, ultrices porttitor, pede. Nullam sit amet nisl elementum elit
convallis malesuada. Phasellus magna sem, semper quis, faucibus ut, rhoncus
non, mi. Duis pellentesque, felis eu adipiscing ullamcorper, odio urna
```

```
consequat arcu, at posuere ante quam non dolor. Lorem ipsum dolor sit amet,
consectetuer adipiscing elit. Duis scelerisque.</p>
<a id="enlace_1" href="#">Ocultar contenidos</a>

<br/>

<p id="contenidos_2">[2] Lorem ipsum dolor sit amet, consectetuer adipiscing
elit. Sed mattis enim vitae orci. Phasellus libero. Maecenas nisl arcu,
consequat congue, commodo nec, commodo ultricies, turpis. Quisque sapien nunc,
posuere vitae, rutrum et, luctus at, pede. Pellentesque massa ante, ornare id,
aliquam vitae, ultrices porttitor, pede. Nullam sit amet nisl elementum elit
convallis malesuada. Phasellus magna sem, semper quis, faucibus ut, rhoncus
non, mi. Duis pellentesque, felis eu adipiscing ullamcorper, odio urna
consequat arcu, at posuere ante quam non dolor. Lorem ipsum dolor sit amet,
consectetuer adipiscing elit. Duis scelerisque.</p>
<a id="enlace_2" href="#">Ocultar contenidos</a>

<br/>
<p id="contenidos_3">[3] Lorem ipsum dolor sit amet, consectetuer adipiscing
elit. Sed mattis enim vitae orci. Phasellus libero. Maecenas nisl arcu,
consequat congue, commodo nec, commodo ultricies, turpis. Quisque sapien nunc,
posuere vitae, rutrum et, luctus at, pede. Pellentesque massa ante, ornare id,
aliquam vitae, ultrices porttitor, pede. Nullam sit amet nisl elementum elit
convallis malesuada. Phasellus magna sem, semper quis, faucibus ut, rhoncus
non, mi. Duis pellentesque, felis eu adipiscing ullamcorper, odio urna
consequat arcu, at posuere ante quam non dolor. Lorem ipsum dolor sit amet,
consectetuer adipiscing elit. Duis scelerisque.</p>
<a id="enlace_3" href="#">Ocultar contenidos</a>

</body>
</html>
```

11.15. Ejercicio 15

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Ejercicio 15 - Información sobre eventos</title>

<style type="text/css">
body {font-family: arial, helvetica;}
#info {width:160px; border:thin solid silver; padding:.5em; position:fixed;}
#info h1 {margin: 0;}
</style>

<script type="text/javascript">
function informacion(elEvento) {
  var evento = elEvento || window.event;
  switch(evento.type) {
    case 'mousemove':
      document.getElementById('info').style.backgroundColor = '#FFFFFF';
      var ie = navigator.userAgent.toLowerCase().indexOf('msie')!=-1;
      var coordenadaXrelativa, coordenadaYrelativa, coordenadaXabsoluta,
```

```
coordenadaYabsoluta;
      if(ie) {
        if(document.documentElement && document.documentElement.scrollTop){
          coordenadaXabsoluta = evento.clientX +
document.documentElement.scrollLeft;
          coordenadaYabsoluta = evento.clientY +
document.documentElement.scrollTop;
        }
        else {
          coordenadaXabsoluta = evento.clientX + document.body.scrollLeft;
          coordenadaYabsoluta = evento.clientY + document.body.scrollTop;
        }
      }
      else {
        coordenadaXabsoluta = evento.pageX;
        coordenadaYabsoluta = evento.pageY;
      }
      coordenadaXrelativa = evento.clientX;
      coordenadaYrelativa = evento.clientY;
      muestraInformacion(['Ratón', 'Navegador ['+coordenadaXrelativa+',
'+coordenadaYrelativa+']', 'Pagina ['+coordenadaXabsoluta+',
'+coordenadaYabsoluta+']']);
      break;
    case 'keypress':
      document.getElementById('info').style.backgroundColor = '#CCE6FF';
      var caracter = evento.charCode || evento.keyCode;
      var letra = String.fromCharCode(caracter);
      var codigo = letra.charCodeAt(0);
      muestraInformacion(['Teclado', 'Carácter ['+letra+']', 'Código
['+codigo+']']);
      break;
    case 'click':
      document.getElementById('info').style.backgroundColor = '#FFFFCC';
      break;
  }
}

function muestraInformacion(mensaje) {
  document.getElementById("info").innerHTML = '<h1>'+mensaje[0]+'</h1>';
  for(var i=1; i<mensaje.length; i++) {
    document.getElementById("info").innerHTML += '<p>'+mensaje[i]+'</p>';
  }
}

document.onmousemove = informacion;
document.onkeypress = informacion;

document.onclick = informacion;

</script>
</head>

<body>

<div id="info"></div>
```

```
<br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/>
<br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/>
<br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/>

</body>
</html>
```

11.16. Ejercicio 16

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Ejercicio 16 - Información sobre eventos</title>

<style type="text/css">
body {font-family: arial, helvetica;}
#info {width:160px; border:thin solid silver; padding:.5em; position:fixed;}
#info h1 {margin: 0;}
</style>

<script type="text/javascript">
function informacion(elEvento) {
  var evento = elEvento || window.event;

  var coordenadaX = evento.clientX;
  var coordenadaY = evento.clientY;

  var dimensiones = tamanoVentanaNavegador();
  var tamanoX = dimensiones[0];
  var tamanoY = dimensiones[1];

  var posicionHorizontal = "";
  var posicionVertical = "";

  if(coordenadaX > tamanoX/2) {
    posicionHorizontal = "derecha";
  }
  else {
    posicionHorizontal = "izquierda";
  }

  if(coordenadaY > tamanoY/2) {
    posicionVertical = "abajo";
  }
  else {
    posicionVertical = "arriba";
  }

  muestraInformacion(['Posicion', posicionHorizontal, posicionVertical]);
}

function muestraInformacion(mensaje) {
```

```
  document.getElementById("info").innerHTML = '<h1>'+mensaje[0]+'</h1>';
  for(var i=1; i<mensaje.length; i++) {
    document.getElementById("info").innerHTML += '<p>'+mensaje[i]+'</p>';
  }
}

function tamanoVentanaNavegador(){
  // Adaptada de http://www.howtocreate.co.uk/tutorials/javascript/browserwindow
  var dimensiones = [];

  if(typeof(window.innerWidth) == 'number') {
    // No es IE
    dimensiones = [window.innerWidth, window.innerHeight];
  } else if(document.documentElement && (document.documentElement.clientWidth
|| document.documentElement.clientHeight)) {
    //IE 6 en modo estandar (no quirks)
    dimensiones = [document.documentElement.clientWidth,
document.documentElement.clientHeight];
  } else if(document.body && (document.body.clientWidth ||
document.body.clientHeight)) {
    //IE en modo quirks
    dimensiones = [document.body.clientWidth, document.body.clientHeight];
  }

  return dimensiones;
}

document.onclick = informacion;

</script>
</head>

<body>

<div id="info"></div>

<br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/>
<br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/>
<br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/><br/>

</body>
</html>
```

11.17. Ejercicio 17

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Ejercicio 17 - Limitar número de caracteres en textarea</title>

<style type="text/css">
body {font-family: arial, helvetica;}
</style>
```

```
<script type="text/javascript">
function limita(elEvento, maximoCaracteres) {
  var elemento = document.getElementById("texto");

  // Obtener la tecla pulsada
  var evento = elEvento || window.event;
  var codigoCaracter = evento.charCode || evento.keyCode;
  // Permitir utilizar las teclas con flecha horizontal
  if(codigoCaracter == 37 || codigoCaracter == 39) {
    return true;
  }

  // Permitir borrar con la tecla Backspace y con la tecla Supr.
  if(codigoCaracter == 8 || codigoCaracter == 46) {
    return true;
  }
  else if(elemento.value.length >= maximoCaracteres ) {
    return false;
  }
  else {
    return true;
  }
}

function actualizaInfo(maximoCaracteres) {
  var elemento = document.getElementById("texto");
  var info = document.getElementById("info");

  if(elemento.value.length >= maximoCaracteres ) {
    info.innerHTML = "Máximo "+maximoCaracteres+" caracteres";
  }
  else {
    info.innerHTML = "Puedes escribir hasta
"+(maximoCaracteres-elemento.value.length)+" caracteres adicionales";
  }
}

</script>
</head>

<body>

<div id="info">Máximo 100 caracteres</div>
<textarea id="texto" onkeypress="return limita(event, 100);"
onkeyup="actualizaInfo(100)" rows="6" cols="30"></textarea>

</body>
</html>
```

11.18. Ejercicio 18

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
```

```
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Ejercicio 18 - Utilidades, Calendario</title>

<link rel="stylesheet" type="text/css" media="all" href="css/
calendar-estilo.css" />

<script type="text/javascript" src="js/calendar.js"></script>
<script type="text/javascript" src="js/calendar-es.js"></script>
<script type="text/javascript" src="js/calendar-setup.js"></script>

</head>

<body>

<p></p>
<input type="hidden" name="fecha" id="fecha" />
<span style="background-color: #ffc; cursor:default; padding:.3em; border:thin
solid #ff0; text-decoration:underline; color: blue;"
onmouseover="this.style.cursor='pointer'; this.style.cursor='hand';
this.style.backgroundColor='#ff8'; this.style.textDecoration='none';"
onmouseout="this.style.backgroundColor='#ffc';
this.style.textDecoration='underline';"
id="fecha_usuario">
Pincha aquí para seleccionar la fecha
</span>


<script type="text/javascript">
Calendar.setup({
  inputField: "fecha",
  ifFormat:   "%d/%m/%Y",
  weekNumbers: false,
  displayArea: "fecha_usuario",
  daFormat:    "%A, %d de %B de %Y"
});
</script>

</body>
</html>
```

11.19. Ejercicio 19

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Ejercicio 19 - Utilidades, Tooltip</title>

<script type="text/javascript" src="js/overlib.js"><!-- overLIB (c) Erik Bosrup
--></script>
<script type="text/javascript">
overlib_pagedefaults(WIDTH,150,FGCOLOR,'#ffffcc',BGCOLOR,'#666666',TEXTFONT,"Arial,
Helvetica, Verdana",TEXTSIZE,".8em",DELAY,250,OFFSETX,15,ABOVE);
```

```
</script>
</head>

<body>

<p>Lorem ipsum dolor sit amet, <a href="#" onmouseover="return overlib('Prueba
de un tooltip básico y muy sencillo.');" onmouseout="return nd();">consectetuer
adipiscing elit</a>. Etiam eget metus. Proin varius auctor tortor. Cras augue
neque, porta vitae, vestibulum nec, pulvinar id, nibh. Fusce in arcu. Duis
vehicula nonummy orci.</p>

<p>Quisque facilisis, lacus vel adipiscing tempor, neque quam tincidunt ante,
sed bibendum velit urna quis ipsum. Aliquam convallis justo at turpis. Sed
iaculis dictum neque. Praesent varius augue quis ligula. Suspendisse
potenti.</p>

<p>Pellentesque habitant morbi tristique senectus et netus et malesuada fames
ac turpis egestas. Duis non nunc at augue ultrices viverra. Pellentesque
tempus. Duis nisi ipsum, condimentum vel, accumsan non, sodales hendrerit,
nisi. Aenean vel velit. Proin at massa.</p>

</body>
</html>
```